RIP Server Technical Reference

Interniche Legacy Document

Version 1.00

Date: 09-May-2017 12:10

Table of Contents

1 Overview

This Technical reference is provided with the InterNiche Routing Information Protocol (RIP) software. The purpose of this document is to provide enough information so that a moderately experienced "C" programmer with a reasonable understanding of TCP/IP protocols can port the InterNiche RIP software to a new environment.

If the RIP code was delivered as part of an InterNiche TCP/IP stack, there is little or nothing to do - the RIP layers were compiled, linked and tested with the IP stack. This manual is intended primarily as an aid to programmers porting InterNiche RIP to a non-InterNiche IP stack.

1.1 Terms and Conventions

In this document, the term "stack", when used without other qualification, means the TCP/IP and related code as ported to an embedded system. "System" refers to your embedded system. "Sockets" refers to the TCP API developed for UNIX at U.C. Berkeley. A "porting engineer" refers to the engineer who is porting the RIP code. An "end user" refers to the person who ultimately ends up using the Engineer's product. "FCS" is an acronym for "First Customer Ship", the point in the software development cycle when the product is declared ready to ship. A "packet" is sequence of bytes sent on network hardware, also known as a "frame" or a datagram".

Names of files, C structures and C routines are displayed as follows: `c_routine()`.

Small samples of source code from C programs is displayed in these boxes:

```
/* C source file - the world's 1 millionth hello program. */
main()
{
   printf("hello world.\n");
}
```

1.2 What RIP Does

Routing Information Protocol (RIP) is used by IP stacks to exchange routing table information. It is intended for use within the IP-based networks.

IP based networks are often organized into a number of networks connected by machines called routers, or gateways. The networks may be point-to-point links (such as modem connections) or more complex networks such as Ethernet. Data is sent from computer to computer in blocks called datagrams. The sender may be able to send the datagram directly to the destination, or indirectly through a router that is nearer to the destination. On large networks (such as the Internet) datagrams may travel though as many as 255 routers during transit from sender to receiver. A computer which sends such a datagram does not need know about all the routers between it and the destination. It only needs to know how to send to the first router, which will find the next, and so on until the last router sends the datagram directly to the destination. As a datagram traverses a large net in this manner, the various routers which forward it are referred to as the datagrams "hops", and the next router in the chain if referred to as the "next hop".

Routing is the method by which a sender (networked computer or router) determines the next hop for forwarding a datagram. When the datagram is sent, the router decides what route to take by referencing a database known as a routing table. RIP is a mechanism for routers to share their routing table information with other routers and computers.

RIP version 2 is the latest standard (RFC 1723), adding important new features and extensions to the earlier standard which is now referred to as RIP version 1 (RFC 1058). InterNiche's RIP software supports both versions.

Some of the features of the InterNiche RIP Source are:

- Supports both RIP1 and RIP2
- Sends triggered updates
- It uses RIP_SPLIT_HORIZON method for neighboring gateways.
- Periodically checks the entries in the RIP table. If the entries have not been updated for RIP_TTL seconds, then the deletion process is started.
- RIP sends RIP broadcasts every few seconds. The exact interval time is set by the define RIP_BROADCAST_INTERVAL seconds
- If it receives a RIP response, then it updates the Route Table.
- If it receives a RIP request, then it replies with proper RIP response.
- Includes statistics & table display routines, which can be integrated with InterNiche's menuing system or any other simple UI. Using RIP->Show Route Table, the contents of the route table can be viewed and updated.
- If the RIP_AUTHENTICATION flag is set, then authentication is done on per interface basis. So if authentication is enabled for a particular interface, then input RIP packets are authenticated and output RIP packets contain an authentication entry. By default there will be no authentication.

1.3 What a Port Is

In the world of portable networking code, the code designer does not know what tasking system, user applications, or interfaces will be supported in the target system. So a "portable" stack is one that's designed with simple, generic interfaces in these areas and a "glue" layer is created which maps this generic interface into the specific interfaces available on the target system. Using the example of sending a packet, the stack would be designed with a generic "send_packet()" call, and to porting engineer would code a "glue" routine to send the packet on the target system's network interface hardware.

Making a stack portable involves minimizing the number of calls which have go across glue routines, and keeping the glue routines simple and therefore easy to implement. The glue routines also need to be well documented. The interfaces to the InterNiche stack have evolved through years of porting to a variety of processors, network media, and tasking systems. Wherever possible we have used standard interfaces (e. g. Sockets, ANSI C library) or included glue routines to illustrate their use.

The bulk of the work in porting a stack is understanding and implementing these glue routines. The InterNiche RIP server has two kinds of glue routines: the first kind is used to interface to the IP layer, and the second kind to manage the RIP databases (tables, etc.).

1.4 Requirements

Before beginning a port, the programmer should ensure that the necessary resources are available in the target environment. Here is a brief summary of services InterNiche RIP needs from the system:

- A timer which ticks at least once a second.
- A non-volatile read/write method for storing database items (e.g. disk or flash memory)
- Memory as described below
- And of course, an IP stack

Operating System Requirements

The RIP server also requires a few basic services from the Operating System. These are listed here:

clock tick	rip_check() needs to be called once a second to send RIP broadcasts or triggered updates and cleanup old RIP routes.
memory access	RIP obtains dynamic memory by calls to the primitives RIP_ALLOC() and RIP_FREE().These can be mapped directly to the standard calloc() and free() library calls. They can also be mapped to a "partition" based system with very little effort.

1.5 RIP Source Directories List

When distributed without InterNiche IP, the sources for RIP are typically sent in a .zip file which file should be unzipped in such a way as to preserve the underlying directory structure. It contains the sources for the RIP implementation as well test and debug capabilities.

2 Step by Step Porting Guide

2.1 Overview of Porting RIP

This section describes the steps needed to port the InterNiche RIP to a new environment. The discussions below generally assume that the stack is being ported to a small or embedded system with a sockets API interface and that a minimal ANSI C library is available.

The recommended steps to getting the server working on your target system are as follows:

1. Copy the portable source files into your development environment.
2. Create your version of `ripport.h` and compile portable sources.
3. Code your glue layers in `ripport.c` and compile.
4. Build a system, test and debug.

Coding Conventions

The following conventions followed in the RIP source code:

- Boolean variables have the values `TRUE` or `FALSE`. Explicit matching should be done in expressions (for example `if (send_bcast_flag == TRUE )`).
- Functions return a value of `SUCCESS` or error number, thus to do error checking the result of a function call should be explicitly compared with SUCCESS (for example, `if (rip_refuse_lookup(fhost) == SUCCESS)`).
- Interface numbers are always 1 based values.

2.2 Source Files List

Before beginning step one, you should be aware of which files in the InterNiche RIP distribution are the "portable" files, and which are not. The portable files are those which should be compiled and used on any target system without modification. The unportable, or "port dependent" files, are those which will need to be replaced or heavily modified for different target systems. The following is a list of RIP source files which should NOT need to be modified in the course of a normal port. If you feel you need to modify one of these files in the course of a routine port, please discuss it with InterNiche's technical support staff first, so we can either suggest an alternative, or modify our sources to reflect the change.

The portable RIP source files. These should not need to be modified.

- `rip.c`
- `ripauth.c`
- `riprefus.c`
- `riperr.c`
- `rip_mod.c`
- `rip.h`

The network (Sockets) glue files:

- ripport.c
- ripport.h
- rip_nt.c

2.3 CPU Architecture

Many of the common port-dependent architectural issues such as byte order and clock ticks are handled by macros in ipport.h and other files in the target directory. See the NicheStack Refernce manual for a discussion of these ssues. The file ripport.h has a few port-dependent macros that are specific to the RIP module. It is important to review these macros line-by-line to determine if modifications are needed for your target.

Memory Allocation

The RIP code allocates and frees memory blocks dynamically as it runs. It uses the macros listed below to do this. If your target system supports standard C calloc() and free(), the macros map directly as follows:

```
#define RIP_ALLOC(size) calloc(1,size)
#define RIP_FREE(ptr) free(ptr)
```

Many RTOS systems do not use calloc() due to performance issues. Generally, they use a system which supports allocations of fixed size "partitions" (blocks) instead. The macros above are designed to support this - the RIP_ALLOC() macro only allocates a single size, (which will vary from target to target). Thus the macros can be mapped to a call to allocate the next largest partition size.

Debugging Aids

dtrap() is a macro called by the RIP code whenever it detects a situation which should not be occurring. The intention is for the dtrap() routine or macro to try to trap to whatever debugger may be in use by the programmer. Think of it as an embedded break point. For most Intel x86 processor debuggers, this can be done with an int 3 opcode. The macro below is effective if your Intel C compiler accepts inline assembly:

```
#define dtrap();   _asm{ int 3 }
```

You may need to experiment with the exact syntax to get it to compile. The stack code will generally continue executing after a dtrap(), but the dtrap()s usually indicate that something is wrong with the port. **No product based on this code should be shipped until the causes of all calls to dtrap() have been eliminated or are understood.** When it comes time to ship code, the dtrap()s can be redefined to a null function to slightly reduce code size.

The next few primitives have the same function and syntax as `printf()`. They have separate names so that they can have their output redirected or be completely disabled independently of each other. The first, `dprintf()`, is used throughout the stack code to print warning messages when something seems to be wrong. This should be mapped to a debugging console or log during development, and generally "ifdefed away" for FCS. The `gio_printf()` call is for printing statistical information from the RIP menus functions. These will certainly be useful during product development, and depending on the nature of the product may be needed in the end user's release. `gio_printf()` uses InterNiche's generic IO mechanism. So the input /output is done with a device, for example, a console or a TELNET session. The `info_printf()` is for printing informational messages, like arrival of a packet, change of metric for a route, etc.

In most ports, these can both be mapped to `printf()` as shown while the product is under development. Note: This example works on Microsoft C, but some compilers will complain about this syntax since it ignores the fact that these names have parameters. You may have to experiment.

```
#define dprintf printf       /* same parms as printf, but works during run time

#define info_printf printf   /* same parms as printf, used to show general info.
```

For some products, it may make sense to define these away before FCS as follows:

```
#define dprintf          /* define to nothing */
#define info_printf      /* define to nothing */
```

Features and Options

Following is a description of all the `#define` options available for RIP. The most important being `RIP_LOCAL_IP`, which decides whether InterNiche's TCP/IP stack is used or not. RFCs 1058 and 1723 describe these features and give examples of when they should be used.

RIP_SUPPORT	If RIP_SUPPORT is defined, then all RIP sources are enabled. That is all .c files of RIP are `#ifdef`'ed for RIP_SUPPORT. Support for RIP module can be enabled/disabled with just one option, that is RIP_SUPPORT. No need to touch makefiles to add/delete RIP files. When RIP is used with InterNiche's TCP/IP stack, then RIP_SUPPORT is enabled/disabled from the file `ipport.h`
RIP_INICHE_IP	If this is defined, then RIP implementation is tightly bound to InterNiche's TCP/IP stack. This affects routing tables and various other things.
RIP_SILENT_PROCESS	If RIP_SILENT_PROCESS is defined, then the RIP implementation is a PASSIVE one. It will not respond to any REQUESTs. But it will update its tables if any RESPONSE arrives.
RIP_SPLIT_HORIZON	If RIP_SPLIT_HORIZION is defined then the SPLIT_HORIZON technique is used to detect and recover from unreachable destinations. What this means is when sending a RIP packet on an interface, for all gateways on that interface, we will have metric=16 (UNREACHABLE)
RIP_POISONED_REVERSE	If RIP_POISONED_REVERSE is defined then the SPLIT_HORIZON_WITH_POISONED_REVERSE technique is used to detect and recover from unreachable destinations. What this means is when sending a RIP packet on an interface, we will not send info about all gateways on that interface.
RIP_ONLY_RIP2	This means that we are only supporting RIP version 2. Other versions (eg RIP1) are not supported.
RIP_ONLY_RIP1	This means that we are only supporting RIP version 1. Other versions (eg RIP2) are not supported.
RIP_RIP1_AND_RIP2	This means that we are supporting RIP1 and RIP2. RIP Versions are a per-interface issue (according to RFC). The decision made here (to use RIP1 or RIP2 or both) will be used for all interfaces. If a particular value is to be hardcoded for an interface, it can be done in `rip_version_init()`, overriding the default values. The receiving and sending versions for an interface can be different. Note that if you are hardcoding RIP_VERSION_1 for interface 2, you will have to initialize the appropriate elements of the receive and send arrays

RIP_AUTHENTICATION	It is valid only for RIP2. If we are supporting RIP2, then this define will decide whether we will have authentication for RIP packets or not. For RIP1, it has no meaning. Defining it for RIP1 will only increase code size.
RIP_USE_MULITCAST	It is valid only for RIP2. If we are supporting RIP2, then this define will decide whether we will send RIP RESPONSES on a MULTICAST address or BROADCAST address. For RIP1, it has no meaning. Defining it for RIP1 will only increase code size.
RIP_DEFAULT_ROUTE	If this is set, then we would allow the use of default routes (If an entries has destination=0.0.0.0 , then the `ipRouteNextHop` field is treated as default gateway)
RIP_TRIGGERED_UPDATES	By default, we should have triggered updates. If this flag is defined, then we will send triggered updates. Otherwise we will not send.
RIP_REFUSE_LOOKUP	If RIP_REFUSE_LOOKUP is defined, then we will allow the use of a REFUSE list. Hosts can be added to this this. We will reject a RIP RESPONSE if it from any host in the REFUSE list.
RIP_SHOW_STATISTICS	If RIP_SHOW_STATISTICS is defined, then the functions which can be used for viewing the statistics are included in the source code.
RIP_ONLY_SUBNETS	If RIP_ONLY_SUBNETS is defined, then only SUBNET routes are stored in the RIP table. So in this case we store entries like 199.86.12.0, 199.88.45.0, 128.9.0.0, 10.0.0.0 in the Route Table.
RIP_NAT_BCAST_CHECK	if RIP_NAT_BCAST_CHECK is defined, then when sending a RIP broadcast (response), we omit all entries which are either broadcast addresses, or reserved network numbers of NAT
RIP_SHOW_ERROR_MSG	If RIP_SHOW_ERROR_MSG is defined, then a description of the error will be shown (along with error number). Otherwise only error number is shown.
RIP_INICHE_RAND	If RIP_INICHE_RAND is defined, then the `rand()` (used to define a random trigger interval) function is provided by InterNiche. If it is not defined, then the system provided `rand()` function is used. This mechanism is provided for targets which don't have rand() implementation. InterNiche implementation is quite trivial, as we can use cticks for this purpose.

2.4 ripport.c

Once you've developed your `ripport.h` file as described in the previous section, the next step is to code the glue layers.

Timers & Multitasking

The following functions use the Operating System specific calls for time-ticks. Again, they work for InterNiche's TCP/IP stack and WinSock. They should be modified if you are using any other TCP/IP stack.

- `rip_start_timer()`
- `rip_check_timeout()`

The other aspect of multitasking is to protect sensitive structures from being corrupted by code re-entry. This is accomplished by two macros which protect critical sections of code. These are named `ENTER_CRIT_SECTION()` and `EXIT_CRIT_SECTION()`. On Intel systems they can usually be defines as follows:

```
#define ENTER_CRIT_SECTION();   {_asm{ pushf };  _asm{ cli } }
#define EXIT_CRIT_SECTION();   _asm{ popf };
```

The examples given are for the DOS port, where simple disabling interrupts for a brief period is sufficient. On a true real-time system, these should be mapped to a mutex.

Transport (UDP) Layer

As supplied RIP includes code to interface with InterNiche's standard sockets or Microsoft WinSock. You need to create the routines listed below if you have another TCP/IP stack.

`rip_udp_init()`	Initialize RIP
`rip_udp_recv()`	Callback routine for received RIP datagrams.
`rip_udp_cleanup()`	Clean up the data structures allocated in rip_udp_init()
`rip_udp_alloc()`	Allocate a buffer for sending a RIP packet
`rip_udp_free()`	Free a buffer allocated in rip_udp_alloc()
`rip_udp_send()`	Send a RIP packet.

2.5 The RIP Static Data Requirements

During initialization, RIP sets many operational parameters in accordance with RFC1058. The table below shows some of the RIP global variables and their default values. The defaults are initially set by defines in `ripport.h`. If you have included the CLI menus for RIP, then they can be modified at initialization time by script commands. (See the NicheStack Reference manual for a description of script files).

Global Variables

`rip_default_ttl`	Default Time To Live for a route in seconds. Default value for this parameter is 180 seconds.
`rip_def_bcast_interval`	Interval for doing periodic broadcasts in seconds. Default value for this parameter is 30 seconds.
`rip_def_deletion_interval`	Interval for which a route should be stored before it is completely removed from the table. If a route has not been updated for `rip_default_ttl` seconds, then the deletion process is started. After `rip_def_deletion_interval`, the route is completely removed from the table. Default value for this parameter is 120 seconds.
`rip_def_trigger_interval`	Whenever triggered updates are to be done, there is a minimun of 1 to `rip_def_trigger_interval` seconds gap between them. This is done to avoid flooding of the network. Default value for this parameter is 5 seconds.
`rip_default_flags`	This variable can be `RIP_SPLIT` or `RIP_POISON`. The integer value for `RIP_SPLIT` is 1 and `RIP_POISON` is 2 in `ripport.h`. So the valid values are 1 or 2. If 0 is specified, then default value is used.
`rip_allow_default_gateways`	If this value is 0 then use of default gateways will be disabled. For non-zero values, use of default gateways will be enabled.

RIP Routes

Routes can be added to the RIP routing table using the "ripaddroute" command. Normally this would be done at initialization time via the "iniche_rc" file. If menu commands are not available and IP_ROUTING is not defined, then the porting engineer will have to provide another means to initialize the routing table.

If `IP_ROUTING` is defined, the RIP will use the routing table "`struct RtMib rt_mib`", which was set up by the IP Routing code. RIP will also use two other key values initialized by the IP Routing code: The maximum number of entries in the routing table, "`rg.max_entries`"), is set to the value of the global "`ipRoutes`", and the number of interfaces, "`rt.num_ifaces`", is set to the return from the function "`ip_num_of_interfaces()`".

If `IP_ROUTING` is defined and RIP is running on a system that with multiple network interfaces, the IP routing code will automatically configure the table with a route for each of the networks over which the IP protocol will run. For example, if the system has two networks cards with network addresses of XX.0.0.0 and YY.0.0.0 then RIP will broadcast RIP responses where the first two routes will be XX.0.0.0 and YY. 0.0.0.

Note: On the XX.0.0.0 network the RIP responses will have a metric of 16 (infinity) for XX.0.0.0 (every node on this network automatically knows how to reach every other node on this network) and a metric of 1 for YY.0.0.0. On the YY.0.0.0 network, the metrics for these two networks will be reversed.

2.6 Testing

Once your `ripport.h` file is set up and your glue layers are coded, compiled, and linked, you are ready to test your RIP. There are two aspects to this - verifying the RIP protocol is processing RIP packets correctly, and testing the IP layer to be sure the Routing information provided by RIP is being applied correctly. The second of these issues (making sure your IP is using routes properly) is beyond the scope of this manual, so the rest of this section is devoted to the former issue.

For most porting engineers , the simplest way of doing this will be with InterNiche's WinSock application. Simply attach your RIP-enabled target system and a Windows PC to the same network segment, start both up, and use the Windows application to send RIP queries to your target system. If you ported the provided statistics routines to your target, verifying that the two RIP hosts are exchanging routing information will be simple.

Most problems which occur at this point have to do with bugs in the implementation of the transport layer. Oversized or mis-routed packets; endian-swapped IP addresses, and errors in sockets semantics are all quite common.

The RIP server, unlike many networking protocols, is quite amenable to source level debugging with breakpoints. Setting a breakpoint on `rip_udp_recv()` will allow you to trace the entire processing of a received RIP packet. Setting a breakpoint on `rip_check()` will allow you to trace the sending of RIP broadcasts, sending of triggered updates, and cleaning up of route entries.

In all cases, a Packet Analyzer is an invaluable tool for debugging this sort of problem. An analyzer will capture on packets on the LAN to which it is attached, and save them for later review. Most support filters, so you can set them to capture only the packets of interest - in this case RIP packets.

3 The RIP User Menu

The RIP comes with portable C code to implement a few simple diagnostic commands on command line interface. The commands can be invaluable both during debugging of the server and to the end user during configuration and runtime. If you do not implement these menu commands as provided, we strongly suggest that some alternative method (i.e. a GUI) be provided to the end user for accessing the same data.

3.1 rip config

Command Name

`rip config - Configure RIP global variables`

Syntax

```
rip config -b <broadcast> -d <deletion> -t <time to live> -z <trigger>
```

Parameters

`-b`	Broadcast interval in seconds
`-d`	Deletion interval in seconds
`-t`	time to live in seconds
`-z`	trigger interval in seconds

Description

This command configures RIP global variables

Notes/Status

- The defaults for these variables were all set according to the recommendations of the RIP specification.

Location

This command is provided by the `RIP` module when `RIP SUPPORT` is defined.

3.2 rip netstat

Command Name

`rip netstat - Displays RIP statistics`

Syntax

```
rip netstat
```

Parameters

None	Command takes no parameters

Description

This command is used to display statistics for RIP.

Location

This command is provided by the `RIP` module when `RIP_SUPPORT` is defined.

3.3 rip ripauth

Command Name

`ripauth - Add/remove/display entries in the RIP authorization table`

Syntax

```
ripauth [{-a | -r} -i <iface> -p <password>]
```

Parameters

	ripauth without parameters will display the current RIP authorization table
`-a`	Add an entry to the RIP authorization table
`-r`	Remove an entry from the RIP authorization table
`-i`	Interface
`-p`	Password

Description

This command is used to add, remove, or display entries in the RIP authorization table

Notes/Status

- The authorization table only exists when RIP_AUTHENTICATION is defined.
- If parameters are entered, then either `'-a'` or `'-r'` is required, and `'-i'` and `'-p'` are both required.

Location

This command is provided by the `RIP` module when `RIP_SUPPORT` and `RIP_AUTHENTICATION` are defined.

3.4 rip riprefuse

Command Name

`riprefuse - Add/remove/display entries in the RIP refuse table`

Syntax

```
riprefuse [{-a | -r} -i <ipaddr>]
```

Parameters

	riprefuse without parameters will display the current RIP refuse table
`-a`	Add an entry to the RIP refuse table
`-r`	Remove an entry from the RIP refuse table
`-i`	IP address

Description

This command is used to add, remove, or display entries in the RIP refuse table

Notes/Status

- The refuse table only exists when RIP_REFUSE_LOOKUP is defined.
- If parameters are entered, then either `'-a'` or `'-r'` is required and `'-i'` is required.

Location

This command is provided by the `RIP` module when `RIP_SUPPORT` and `RIP_REFUSE_LOOKUP` are defined.

3.5 rip riproute

Command Name

`riproute - add/remove/display routes`

Syntax

```
riproute [{-a | -r} -d <dest> -i <iface> -s <subnetmask> [-g <gw>] [-m
<metric>] [-t <ttl>] [-f [PRIVATE | TRIGGER] [-p <proxy>] ]
```

Parameters

	riproute without parameters displays the RIP routing table
`-a`	Add entry to RIP routing table
`-r`	Remove entry from RIP routing table
`-d`	Destination IP address
`-i`	Interface
`-s`	Subnet mask
`-g`	Gateway IP addr
`-m`	Metric
`-t`	Time to live (TTL)
`-f`	Flag
`-p`	Proxy IP address

Description

This command is used to set up entries in the routing table in order to be used by RIP.

RIP is a protocol that exchanges routing information between routers.

Notes/Status

- If IP_ROUTING is defined, the routing table will automatically be filled with all basic basic routing entries: an entry for each routable interface on the local system plus those learned from connected routers. RFC 1723 describes special cases where you may wish to add additional routes to these automatically generated routes.
- This command currently only supports IPv4 routes.
- If parameters are entered, then either `'-a'` or `'-r'` is required, and `'-d'`, `'-i'`, and `'-s'` are all required.
- Unless subnetting should be used for this entry, the value of the subnet mask must identify the full network portion of the address.
- For the -f operand, PRIVATE protects the entry from deletion.
- For the -f operand, TRIGGER is the default. It means that an RIP response message will be sent immediately, whenever there is a change in the metric for any entry in the routing table.
- For the -m operand, the metric is the "cost" (typically number of hops) to get to the given node. Note: RIP will substitute "16" (infinity) for this metric in messages sent on networks for which the entry should not be used (split horizon principle).
- The interface identifier is one-based (and not zero-based).
- After TTL seconds, an entry that has not been renewed will be marked as unusable (metric = 16). Then following the "rip_def_deletion_interval", the entry will be removed.

Location

This command is provided by the `RIP` module when `RIP_SUPPORT` is defined.

4 Port Provided Functions

The functions described in this section must be provided by the porting programmer as part of the porting the InterNiche RIP. The Windows reference port can be referenced for examples. In you are using the InterNiche IP stack, many for these functions are already provided therein.

4.1 General Functions

dtrap

Name

`dtrap()`

Syntax

```
void dtrap(void);
```

Parameters

None

Description

This primitive is intend to hook a debugger whenever it is called. See the detailed description in the Debugging Aids section of this document.

Returns

Usually nothing, depends on user modifications.

dprintf

Name

`dprintf()`

`stat_printf()`

`info_printf()`

Syntax

```
void dprintf(char *, ...);

void stat_printf(char *, ...);

void info_printf(char *,...);
```

Parameters

None

Description

These routines are functionally the same as `printf()`. They are called by the stack code to inform the programmer or end user of system status. `dprintf()` prints error warnings during runtime, `stat_printf()` is used to display statistics and `info_printf()` is used to display informational messages. For example, `dprintf()` would be used to display errors, `stat_printf()` for showing statistics via the menu interface, and `info_printf()` to display information about processing that happens in the background (like arrival of a packet, change of a RIP metric, etc.).

ENTER_CRIT_SECTION

Name

`ENTER_CRIT_SECTION()`

`EXIT_CRIT_SECTION()`

Syntax

```
void ENTER_CRIT_SECTION (void);

void EXIT_CRIT_SECTION (void);
```

Parameters

None

Description

These two primitives should be designed to be paired around sections of code that must not be interrupted or pre-empted. Generally these simply need to disable and re-enable interrupts. Only the definitions are given here; for examples see the source code. The stack source code always pairs these two in the same routines.

4.2 RIP IP Layer Interface

For non-InterNiche TCP/IP stacks, RIP needs to have the following support from the IP layer. Providing these is part of the porting process.

ip_is_my_addr

Name

`ip_is_my_addr()`

Syntax

```
int ip_is_my_addr(u_long target);
```

Parameters

IP address to be checked

Description

This function searches all interfaces to see if the target address matches the address of any of those. If a match occurs it returns `SUCCESS`, otherwise it returns `IP_NOT_MY_ADDR`.

Returns

`SUCCESS` or the error `IP_NOT_MY_ADDR`.

ip_num_of_interfaces

Name

`ip_num_of_interfaces()`

Syntax

```
int ip_num_of_interfaces();
```

Parameters

None

Description

This function is returns the number of network interfaces present in the device.

Returns

Numbers of interfaces present in the device.

ip_is_loopback_iface

Name

`ip_is_loopback_iface()`

Syntax

```
int ip_is_loopback_iface(int iface);
```

Parameters

Interface to be checked.

Description

This function determines if the interface is an IP loopback interface or not. In RIP, it is called make sure that we don't send RIP broadcasts on loopback interface. Returns SUCCESS if the interface is a loopback interface, FAILURE otherwise.

Returns

SUCCESS or FAILURE.

ip_get_iface_ipaddr

Name

`ip_get_iface_ipaddr()`

Syntax

```
ip_addr ip_get_iface_ipaddr(int iface);
```

Parameters

Interface number

Description

This function returns the IP address for a particular interface.

Returns

The IP address of the interface.

ip_get_iface_bcastaddr

Name

`ip_get_iface_bcastaddr()`

Syntax

```
ip_addr ip_get_iface_bcastaddr(int iface);
```

Parameters

Interface number

Description

This function returns the broadcast IP address for a particular interface.

Returns

Broadcast address of the interface.

4.3 RIP Transport Layer Interface

rip_udp_init

Name

`rip_udp_init()`

Syntax

```
int rip_udp_init();
```

Parameters

None

Description

This call does the initialization so that RIP packets can be received via `rip_udp_recv()`.

Returns

Returns SUCCESS (0) if everything went OK, else returns a non-zero error code.

rip_udp_recv

Name

`rip_udp_recv()`

Syntax

```
int rip_udp_recv();
```

Parameters

Pointer to the received packet
Length of received data
Source port of received packet
Source IP address of received packet
Interface on which the packet arrived

Description

`rip_udp_recv()` is a callback function. It uses five parameters about the received packet to make a call to `rip_process_rcvd_pkt()`. The function as described here is used with the InterNiche's TCP /IP stack. The structure PACKET contains the data that is needed to call `rip_process_rcvd_pkt()`.

Returns

Returns SUCCESS (0) if everything went OK, else returns a non-zero error code.

rip_udp_cleanup

Name

`rip_udp_cleanup()`

Syntax

```
int rip_udp_cleanup();
```

Parameters

None

Description

This call cleans up the data structures allocated in `rip_udp_init()`.

Returns

Returns SUCCESS (0) if everything went OK, else returns a non-zero error code.

rip_udp_alloc

Name

`rip_udp_alloc()`

Syntax

```
int rip_udp_alloc();
```

Parameters

Size of memory block (IN)
Pointer to `RIP_VIRTUAL_PKT` (OUT)

Description

This function allocates memory for a RIP packet and forms a `RIP_VIRTUAL_PKT` with it. `RIP_VIRTUAL_PKT` is used to store the buffer and length for the packet. It is assumed that the second argument (pointer to `RIP_VIRTUAL_PKT`) points to a valid structure (already allocated structure).

Returns

Returns pointer to `RIP_VIRTUAL_PKT` or `NULL`.

rip_udp_free

Name

`rip_udp_free()`

Syntax

```
int rip_udp_free();
```

Parameters

Pointer to RIP_VIRTUAL_PKT (OUT)

Description

This function frees the memory that was allocated using `rip_udp_alloc()`. The argument to this function is a pointer to `RIP_VIRTUAL_PKT` which was populated using `rip_udp_alloc()`.

Returns

Returns pointer to `RIP_VIRTUAL_PKT` or `NULL`.

rip_udp_send

Name

`rip_udp_send()`

Syntax

```
int rip_udp_send();
```

Parameters

Destination port for the packet (IN)
Source port for the packet (IN)
IP address of the destination (recipient of this packet) (IN)
Pointer to `RIP_VIRTUAL_PKT` (packet to be sent) (IN)

Description

This function sends a RIP packet on the network.

Returns

Returns `SUCCESS` (0) if everything went OK, else returns a non-zero error code.

4.4 RIP Table Manipulation Functions

RIP exports makes several routines available for the Porting Engineer to manage with the internal routing table. The route table, authentication table, and refuse table entries can be added or deleted from these tables using the following functions.

rip_route_add

Name

`rip_route_add()`

Syntax

```
RTMIB rip_route_add(ip_addr dest, ip_addr subnet_mask, ip_addr gateway, int
iface, u_long rip_metric, int interval, int flags,u_long proxy_route,
u_short route_tag);
```

Parameters

Destination IP Address
Subnet mask
New Gateway
Interface Number (1 based value)
Metric
Interval (of timeout)
Flags
Proxy Route
Route Tag

Description

This function adds an entry to the RIP route table.

Returns

Returns pointer to entry which has been added. Returns `NULL` if it could not add an entry.

rip_route_delete

Name

`rip_route_delete()`

Syntax

```
int rip_route_delete(RTMIB rp);
```

Parameters

Pointer to entry to be deleted

Description

This function deletes an entry from the RIP route table.

Returns

Returns SUCCESS (0) if everything went OK, else returns a non-zero error code.

rip_auth_add

Name

`rip_auth_add()`

Syntax

```
int rip_auth_add(int iface, u_short auth_type, char * auth_password);
```

Parameters

Interface Number (IN) (1 based value)
Authentication type (IN)
Authentication Password (IN)

Description

This function adds authentication details for a particular interface. We have assumed that there can be one password per interface.

Returns

Returns SUCCESS (0) if everything went OK, else returns a non-zero error code.

rip_auth_delete

Name

`rip_auth_delete()`

Syntax

```
int rip_auth_delete(int iface);
```

Parameters

Interface Number (IN) (1 based value)

Description

This function deletes authentication entry for an interface.

Returns

Returns SUCCESS (0) if everything went OK, else returns a non-zero error code.

rip_refuse_add

Name

`rip_refuse_add()`

Syntax

```
int rip_refuse_add(ip_addr addr);
```

Parameters

IP Address to be added to Refuse List.

Description

This function adds an entry to the array `refuse_list`. This list is used to maintain list of hosts from which we will not accept RIP information.

Returns

Returns SUCCESS (0) if everything went OK, else returns a non-zero error code.

rip_refuse_delete

Name

`rip_refuse_delete()`

Syntax

```
int rip_refuse_delete(ip_addr addr);
```

Parameters

IP Address to be deleted from Refuse List

Description

This function deletes an entry from the array `refuse_list`.

Returns

Returns SUCCESS (0) if everything went OK, else returns a non-zero error code.