QNX® Neutrino® RTOS

Technical Note for NVIDIA Customers

BlackBerry | QNX

BlackBerry QNX
1001 Farrar Road
Ottawa, Ontario
K2K 0B3
Canada

Voice: +1 613 591-0931
Fax: +1 613 591-3579
Email: info@qnx.com
Web: http://www.qnx.com/

Electronic edition published: March 16, 2021

Contents

About This Technote

This technote provides customized versions of the following reference pages:

dlopen()

Open a shared library

io-pkt-v4-hc, io-pkt-v6-hc

Networking manager

pci_device_reset()

Reset a PCI device

PCIe

Allocate-once policy for device interrupts

ptpd, ptpd-avb

Precision Time Protocol daemon

shutdown

Shut down and reboot the system

You should refer to these entries instead of the ones in the *Utilities Reference* and the *PCI Server User's Guide*.

Typographical conventions

Throughout this manual, we use certain typographical conventions to distinguish technical terms. In general, the conventions we use conform to those found in IEEE POSIX publications.

The following table summarizes our conventions:

Reference	Example
Code examples	`if( stream == NULL)`
Command options	`-lR`
Commands	`make`
Constants	NULL
Data types	`unsigned short`
Environment variables	*PATH*
File and pathnames	**/dev/null**
Function names	*exit()*
Keyboard chords	**Ctrl–Alt–Delete**
Keyboard input	`Username`
Keyboard keys	**Enter**
Program output	`login:`
Variable names	*stdin*
Parameters	*parm1*
User-interface components	**Navigator**
Window title	**Options**

We use an arrow in directions for accessing menu items, like this:

You'll find the Other... menu item under **Perspective** → **Show View**.

We use notes, cautions, and warnings to highlight important messages:

Notes point out something important or useful.

CAUTION: Cautions tell you about commands or procedures that may have unwanted or undesirable side effects.

DANGER: Warnings tell you about commands or procedures that could be dangerous to your files, your hardware, or even yourself.

Note to Windows users

In our documentation, we typically use a forward slash (/) as a delimiter in pathnames, including those pointing to Windows files. We also generally follow POSIX/UNIX filesystem conventions.

Technical support

Technical assistance is available for all supported products.

To obtain technical support for any QNX product, visit the Support area on our website (www.qnx.com). You'll find a wide range of support options, including community forums.

Chapter 1
dlopen()

Open a shared library

Synopsis:

```
#include <dlfcn.h>

void * dlopen( const char * pathname,
               int mode );
```

Arguments:

pathname

NULL, or the path to the shared library that you want to access.

mode

Flags that control how *dlopen()* operates. POSIX defines these bits:

- RTLD_LAZY
- RTLD_NOW
- RTLD_GLOBAL
- RTLD_LOCAL

The following bits are Unix extensions:

- RTLD_NOLOAD
- RTLD_GROUP
- RTLD_WORLD
- RTLD_NODELETE

The following bits are QNX Neutrino extensions:

- RTLD_NOSHARE
- RTLD_LAZYLOAD

For more information, see "The *mode*," below.

Library:

libc

Use the `-l c` option to `qcc` to link against this library. This library is usually included automatically.

Description:

The *dlopen()* function gives you direct access to the dynamic linking facilities by making the shared library specified in *pathname* available to the calling process. It returns a handle that you can use in subsequent calls to *dlsym()* and *dlclose()*.

- To successfully call this function, your process must have the PROCMGR_AID_PROT_EXEC and PROCMGR_AID_MAP_FIXED abilities enabled. For more information, see *procmgr_ability()*.
- Any child process that you created also must have the PROCMGR_AID_PROT_EXEC and PROCMGR_AID_MAP_FIXED abilities enabled to use *dlopen()*.
- If your process has any privileged abilities enabled, then the shared library you're trying to open must be trusted. For more information, see the description of PROT_EXEC in the entry for *mmap()*, as well as the entry for `pathtrust` in the *Utilities Reference*.
- The *dlopen()* function is available only to a dynamically-linked process. A statically-linked process (one where **libc** is linked statically) can't call *dlopen()* because a statically-linked executable:
  - doesn't export any of its symbols
  - can't export the required structure for libraries to link against
  - can't fill structures at startup needed to load subsequent shared objects

Any dependencies recorded within *pathname* are loaded as part of the *dlopen()* call. These dependencies are searched in load-order to locate any additional dependencies. This process continues until all of the dependencies for *pathname* have been satisfied. This dependency tree is called a *group*.

If *pathname* is NULL, *dlopen()* provides a handle to the running process's global symbol object. This provides access to the symbols from the original program image file, the dependencies it loaded at startup, plus any objects opened with *dlopen()* calls using the RTLD_GLOBAL flag. This set of symbols can change dynamically if the application subsequently calls *dlopen()* using RTLD_GLOBAL.

You can use *dlopen()* any number of times to open objects whose names resolve to the same absolute or relative path name; the object is loaded into the process's address space only once.

In order to find the shared objects, *dlopen()* searches the following, in this order:

- the runtime library search path that was set using the `-rpath` option to `ld` (see the *Utilities Reference*) when the binary was linked
- directories specified by the *LD_LIBRARY_PATH* environment variable
- directories specified by the _CS_LIBPATH configuration string

For security reasons, the runtime linker unsets *LD_LIBRARY_PATH* if the binary has the setuid bit set.

The above directories are set as follows:

- The *LD_LIBRARY_PATH* environment variable is generally set up by a startup script, either in the boot image or in a secondary script. It isn't part of any default environment. Each *dlopen()* call

reads back its process's *LD_LIBRARY_PATH* and honors any changes you make to this variable at runtime.

- _CS_LIBPATH is populated by the kernel, and the default value is based on the *LD_LIBRARY_PATH* value of the `procnto` command line in the boot image. Note that you can use `setconf` to set this configuration string. For example:

  ```
  setconf _CS_LIBPATH /usr/lib:/lib:/lib/dll
  ```

  Unlike with *LD_LIBRARY_PATH*, *dlopen()* doesn't reread _CS_LIBPATH, so changes made to this string at runtime aren't recognized.

When loading shared objects, the application should open a specific version instead of relying on the version pointed to by a symbolic link.

The *mode*

The *mode* argument indicates how *dlopen()* operates on *pathname* when handling relocations, and controls the visibility of symbols found in *pathname* and its dependencies.

The *mode* argument is a bitwise-OR of the constants described below. Note that the relocation and visibility constants are mutually exclusive.

Relocation

When you load an object by calling *dlopen()*, the object may contain references to symbols whose addresses aren't known until the object has been loaded; these references must be relocated before accessing the symbols. The *mode* controls when relocations take place, and can be one of:

RTLD_LAZY

(QNX Neutrino 6.5.0 and later) References to data symbols are relocated when the object is loaded. References to functions aren't relocated until that function is invoked. This improves performance by preventing unnecessary relocations.

RTLD_NOW

All references are relocated when the object is loaded. This may waste cycles if relocations are performed for functions that never get called, but this behavior could be useful for applications that need to know that all symbols referenced during execution are available as soon as the object is loaded.

Visibility

The following *mode* bits determine the scope of visibility for symbols loaded with *dlopen()*:

RTLD_GLOBAL

Make the object's global symbols available to any other object that's opened later with RTLD_WORLD. Symbol lookup using `dlopen( 0,` *mode* `)` and an associated *dlsym()* are also able to find the object's symbols.

RTLD_LOCAL

Make the object's global symbols available only to objects in the same group.

RTLD_LAZYLOAD

Open the shared object and form its resolution scope by appending its immediate dependencies. This is different from the normal resolution scope, which is formed by appending the whole dependency tree in breadth-first order. For more information, see "Lazy loading" in the Compiling and Debugging chapter of the QNX Neutrino *Programmer's Guide*.

The program's image and any objects loaded at program startup have a *mode* of RTLD_GLOBAL; the default *mode* for objects acquired with *dlopen()* is RTLD_LOCAL. A local object may be part of the dependencies for more than one group; any object with a RTLD_LOCAL *mode* referenced as a dependency of an object with a RTLD_GLOBAL *mode* is promoted to RTLD_GLOBAL.

Objects loaded with *dlopen()* that require relocations against global symbols can reference the symbols in any RTLD_GLOBAL object.

Symbol scope

You can OR the *mode* with the following values to affect the symbol scope:

RTLD_GROUP

Only symbols from the associated group are available. All dependencies between group members must be satisfied by the objects in the group.

RTLD_WORLD

Only symbols from RTLD_GLOBAL objects are available.

If you don't specify either of these values, *dlopen()* uses RTLD_WORLD | RTLD_GROUP.

If you specify RTLD_WORLD without RTLD_GROUP, *dlopen()* doesn't load any of the DLL's dependencies.

Other flags

The following flags provide additional capabilities:

RTLD_NODELETE

Don't delete the specified object from the address space as part of a call to *dlclose()*.

RTLD_NOLOAD

Don't load the specified object, but return a valid handle if if the object already exists as part of the process address space. You can specify addition modes, which are ORed with the present mode of the object and its dependencies. This flag gives you a means of querying the presence or promoting the modes of an existing dependency.

RTLD_NOSHARE

Don't share the specified object. This flag forces the loading of multiple instances of libraries.

Symbol resolution

When resolving the symbols in the shared object, the runtime linker searches for them in the dynamic symbol table using the following order:

By default:

1. main executable
2. the shared object being loaded
3. objects specified by the *LD_PRELOAD* environment variable
4. all other loaded shared objects that were loaded with the RTLD_GLOBAL flag

When `-Bsymbolic` is specified:

1. the shared object being loaded
2. main executable
3. objects specified by the *LD_PRELOAD* environment variable
4. all other loaded shared objects that were loaded with the RTLD_GLOBAL flag

For security reasons, the runtime linker unsets *LD_PRELOAD* if the binary has the setuid bit set.

For executables, the dynamic symbol table typically contains only those symbols that are known to be needed by any shared libraries. This is determined by the linker when the executable is linked against a shared library.

Since you don't link your executable against a shared object that you load with *dlopen()*, the linker can't determine which executable symbols need to be made available to the shared object.

If your shared object needs to resolve symbols in the executable, then you may force the linker to make *all* of the symbols in the executable available for dynamic linking by specifying the `-E` linker option. For example:

```
qcc -Vgcc_ntox86 -Wl,-E -o main main.o
```

Shared objects always place all their symbols in dynamic symbol tables, so this option isn't needed when linking a shared object.

Returns:

A handle to the object, or NULL if an error occurs.

Don't interpret the value of this handle in any way. For example, if you open the same object repeatedly, don't assume that *dlopen()* returns the same handle.

Errors:

If an error occurs, more detailed diagnostic information is available from *dlerror()*.

Environment variables:

- The runtime linker gets the values of the following environment variables only when the process is loaded:
  - *DL_DEBUG*
  - *LD_PRELOAD*
  - *LD_TRAP_ON_ERROR*

  The *dlopen()* function examines *LD_LIBRARY_PATH* every time you call it.
- For security reasons, the runtime linker unsets *DL_DEBUG*, *LD_DEBUG*, *LD_DEBUG_OUTPUT*, *LD_LIBRARY_PATH*, and *LD_PRELOAD* if the binary has the setuid bit set.

DL_DEBUG

If this environment variable is set, the shared library loader displays debugging information about the libraries as they're opened. The value can be a comma-separated list of the following:

- `all` — display all debug messages.
- `help` — display a help message, and then exit.
- `reloc` — display relocation processing messages.
- `libs` — display information about shared objects being opened.
- `statistics` — display runtime linker statistics.
- `lazyload` — print lazy-load debug messages.
- `debug` — print various runtime linker debug messages.

A value of 1 (one) is the same as `all`.

LD_BIND_NOW

Affects lazy-load dependencies due to full symbol resolution. Typically, it forces the loading of all lazy-load dependencies (until all symbols have been resolved).

LD_DEBUG

A synonym for *DL_DEBUG*. If you set both *DL_DEBUG* and *LD_DEBUG*, then *DL_DEBUG* takes precedence.

LD_DEBUG_OUTPUT

The name of a file in which the runtime linker writes its output. By default, output is written to *stderr*.

LD_LIBRARY_PATH

A colon-separated list of directories to search for shared libraries. Blank entries in the list are treated as ".", meaning they refer to the current working directory. This can include the following cases:

- The entire list is blank (i.e., `LD_LIBRARY_PATH=""` was specified), which is different from the variable being unset.
- The list has a colon at the beginning (i.e., the first entry is blank), or it has two consecutive colons (i.e., the entry in between is blank).

LD_PRELOAD

A list of full paths to the shared libraries on an ELF system that you want to load before loading other libraries. Use a colon (`:`) to separate the libraries in this list. You can use this environment variable to add or change functionality when you run a program.

LD_TRAP_ON_ERROR

If this environment variable is set, the runtime linker faults instead of exiting on fatal errors, so that you can examine the core file that's generated.

Classification:

POSIX 1003.1

Safety:	
Cancellation point	Yes
Interrupt handler	No
Signal handler	No
Thread	Yes

Caveats:

Some symbols defined in executables or shared objects might not be available to the runtime linker. The symbol table created by `ld` for use by the runtime linker might contain a subset of the symbols defined in the object.

Chapter 2
io-pkt-v4-hc, io-pkt-v6-hc

Networking manager

Syntax:

```
io-pkt-variant [-DIS] [-d driver [driver_options]] [-i instance] [-P priority]
               [-p protocol [protocol_options]] [-q interval] [-r priority]
               [-t threads] [-U string] [-u [path_type],uid:gid,mode]
               [-w interval]
```

where *variant* is `v4-hc` or `v6-hc`.

Runs on:

QNX Neutrino

Options:

`-D`

(QNX Neutrino 7.0 or later) Run the resource manager/protocol layer stack context in a dedicated POSIX thread. By default this is off. This option can offer a performance improvement if you're sending and receiving TCP/IP traffic from applications on a multicore system.

`-d` *driver* [*driver_options*]

Start the specified `devnp-*` driver:

- You can specify *driver* without the `devnp-` prefix or the `.so` extension. For example, to start the **devnp-e1000.so** driver, specify `-d e1000`.
- If you want to load a specific version of a driver, specify the full path of the module (e.g., **/lib/dll/devnp-e1000.so**).

The *driver_options* argument is a list of driver-specific options that the stack passes to the driver.

Use commas, not spaces, to separate the options.

The stack processes various driver options; for more information, see "Generic driver options," below.

`-I`

(QNX Neutrino 7.0 or later) Don't acquire _NTO_TCTL_IO_PRIV in this instance of `io-pkt` (see *ThreadCtl()* in the *C Library Reference*). With this option enabled, drivers loaded into `io-pkt` can't perform any operation that requires _NTO_TCTL_IO_PRIV, and they can't queue interrupts in an ISR.

-i ***instance***

The stack instance number, which is useful if you're running multiple instances of `io-pkt`. The `io-pkt` manager will service mount requests of type `io-pkt`*X*, where *X* is the instance number. For example:

```
io-pkt-v4-hc -i1 -ptcpip prefix=/alt
mount -Tio-pkt1 /lib/dll/devnp-abc100.so
```

-P ***priority***

The priority to use for `io-pkt`'s main thread. The default is 21.

-p ***protocol*** **[***protocol_options***]**

The protocol to start, followed by a list of protocol-specific options.

Use commas, not spaces, to separate the options.

The available protocols include:

Protocol	Module
`pf-v4`	**lsm-pf-v4.so** (for use with `io-pkt-v4-hc`)
`pf-v6`	**lsm-pf-v6.so** (for use with `io-pkt-v6-hc`)
`qnet`	**lsm-qnet.so**
`slip`	**lsm-slip.so** (for use with `io-pkt-v4-hc`)
`tcpip`	The stack includes TCP/IP; you need to specify this protocol only if you want to pass additional parameters (e.g., `prefix=`) to it. For more information about the options, see below.

-q ***interval***

(QNX Neutrino 7.0 or later) Set up the *quiesce_all()* watchdog. If *unquiesce_all()* isn't called within *interval* (1–5) seconds, `io-pkt` panics. The watchdog is disabled by default.

-r ***priority***

(QNX Neutrino 7.0 or later) The priority of the watchdog thread, which should be higher than the other `io-pkt` and socket application priorities. The default is 22.

-S

Don't register a SIGSEGV handler to quiesce the hardware if a segmentation violation occurs. This can help with debugging if it isn't possible to get a backtrace to the original code that generated the SIGSEGV through the signal handler.

-t ***threads***

The number of processing threads to create. By default, one thread is created per CPU. These threads are the packet-processing threads that operate at Layer2 and may become the stack thread. For more information, see the Overview chapter of the QNX Neutrino Core Networking *User's Guide*.

-U ***string***

(QNX Neutrino 7.0 or later) Specify the user and groups that `io-pkt` should drop to when you tell it to stop running as **root** (see below). The *string* can be in one of these forms:

- *uid*[:*gid*[,*sup_gid*]*]
- *user_name*[,*sup_gid*]*

The default is `99:99,120`. In the second form, the primary group is the one specified for *user_name* in **/etc/passwd**.

The `io-pkt` manager drops to the user specified with the `-U` option when you issue the following `sysctl` command:

```
sysctl -w qnx.kern.droproot=value
```

The *value* is a hexadecimal number whose bits indicate which abilities `io-pkt` should keep, or 0 if you want `io-pkt` to continue to run as **root**. The QNX_DROPROOT_* flags are defined in **<sys/iopkt_ability.h>**:

Constant	Value	Ability
QNX_DROPROOT_STD	0x0001	Drop **root** without keeping any additional abilities (keep the "standard" ones listed below)
QNX_DROPROOT_INTERRUPT	0x0002	PROCMGR_AID_INTERRUPT
QNX_DROPROOT_CONNECTION	0x0004	PROCMGR_AID_CONNECTION
QNX_DROPROOT_TIMER	0x0008	PROCMGR_AID_TIMER
QNX_DROPROOT_PROT_EXEC	0x0010	PROCMGR_AID_PROT_EXEC
QNX_DROPROOT_PATHSPACE	0x0020	Not used; `io-pkt` keeps PROCMGR_AID_PATHSPACE by default
QNX_DROPROOT_QNET	0x0040	PROCMGR_AID_QNET
QNX_DROPROOT_PUBLIC_CHANNEL	0x0080	PROCMGR_AID_PUBLIC_CHANNEL

By default, `io-pkt` retains the following abilities:

- IOFUNC_ABILITY_DUP
- IOFUNC_ABILITY_EXEC

- IOFUNC_ABILITY_READ
- PROCMGR_AID_KEYDATA
- PROCMGR_AID_MEM_PHYS
- PROCMGR_AID_PATHSPACE
- PROCMGR_AID_PRIORITY

For more information about abilities, see the entry for *procmgr_ability()* in the QNX Neutrino *C Library Reference*.

-u [*path_type*],*uid*:*gid*,*mode*

Specify the user ID, group ID, and access mode for the device files created by `io-pkt`. The *uid* and *gid* parameters can either be their numeric values, or they can be a user name and group name. The *path_type* could be any of the following:

- `socket`
- `bpf`
- `pf`
- `config`
- `tun`
- `tap`
- `llmcast`
- `nraw`
- `slip`
- `pppmgr`
- `all`

The `all` type allows the user to specify the *uid*:*gid*,*mode* combination for all of the `io-pkt` device files.

In general, this option can be specified multiple times, for example:

```
-u all,user1:iopkt,0660 -u bpf,user2:user2,0600
```

This example would set all device nodes to be owned by `user1` and to belong to the group `iopkt`. These device nodes would be accessible only by `user1` and members of the `iopkt` group, with the exception of the `bpf` interface nodes which would be accessible only by `user2`.

Specifying `-u socket,*` or `-u all,*` will also enable permission checking on client accesses through the socket API.

-w *interval*

(QNX Neutrino 7.0 or later) Set up the `io-pkt` watchdog. If the *hardclock_ticks* variable doesn't increase in *interval* (1–5) seconds, `io-pkt` panics. The watchdog is disabled by default.

TCP/IP options

If you specify the `-p tcpip` protocol, the *protocol_options* list can consist of one or more of the following, separated by commas without whitespace:

bigpage_strict

By default, the value of the `pagesize` option is used only for the `mbuf` and cluster pools; the other pools are of size `sysconf(_SC_PAGESIZE)`. If you specify this option, the value of the `pagesize` option is used for all pools.

bigstack[=*size*]

Use a larger stack size for the loading of drivers. Without this option, drivers are loaded using the default `stacksize` stack. With this option, a new larger stack is used. The default size of the larger stack is 128 KB, but you can specify a size, in bytes.

cache=*0*

Disable the caching of packet buffers. This should be needed only as a debugging facility.

confstr_monitor

Monitor changes to configuration strings, in particular CS_HOSTNAME. By default, `io-pkt` gets the hostname once at startup.

fastforward=*X*

Enable (1) or disable (0) fastforwarding path. This is useful for gateways. This option enables, and is enabled by, `forward`; to enable only `forward`, specify `forward,fastforward=0`.

forward

Enable forwarding of IPv4 packets between interfaces; this enables `fastforward` by default. The default is off.

forward6

(`io-pkt-v6-hc` only) Enable forwarding of IPv6 packets between interfaces; off by default.

ipsec

Enable IPsec support; off by default.

mbuf_cache=*X*

As `mbufs` are freed after use, rather than returning them to the internal pool for general consumption, up to *X* `mbufs` are cached per thread to allow quicker retrieval on the next allocation.

mtag_cache=*X*

As `mtags` are freed after use, rather than returning them to the internal pool for general consumption, up to *X* `mtags` are cached per thread to allow quicker retrieval on the next allocation.

This applies only to tags of 16 bytes or less. The default number of tags cached is 128.

mclbytes=*size*

The `mbuf` cluster size. A *cluster* is the largest amount of contiguous memory used by an `mbuf`. If the MTU is larger than a cluster, multiple clusters are used to hold the packet. The default and minimum cluster size is 2 KB (to fit a standard 1500-byte Ethernet packet); the maximum is 64 KB or the value of the `pagesize` option, whichever is smaller. This value is rounded down to the nearest power of two.

Specifying the cluster size can improve performance; for more information, see "Jumbo packets and hardware checksumming" in the Network Drivers chapter of the QNX Neutrino Core Networking *User's Guide*.

mfib_gid_map=*string*

Specify a semicolon-separated mapping of GIDs to Forwarding Information Bases (FIBs). For example, to map SGID 750 and 760 to FIBs 1 and 2, respectively, specify:

```
mfib_gid_map=750_1;760_2
```

num_pool_cache=*X*

The number of pool caches created for `mbuf` and cluster pools. The default is 1 for each pool; the maximum is 20.

num_tap_interface=*X*

The maximum number of TAP interfaces that can be created. The default is 5. You can create TAP interfaces by either the **/dev/tap** cloning device, or via `ifconfig`'s `create` command.

num_tun_control_interface=*X*

The number of TUN control interfaces (e.g., **/dev/tun0**) to create when `io-pkt` starts. The default is 4.

You can create TUN device interfaces (as listed by the `ifconfig` utility) by either opening a precreated control interface or using `ifconfig create`. The only way to create TUN device interfaces beyond the `num_tun_control_interface` number is to use `ifconfig create`.

pagesize=*X*

The smallest amount of data allocated each time for the mbuf and cluster memory pools, or all pools if you specify `bigpage_strict`. This quantum is then carved into chunks of varying size, depending on the pool.

The default value is 128 KB, the maximum is 16 MB, and the minimum is `sysconf(_SC_PAGESIZE)`. This value is rounded down to the nearest power of 2. This value also sets the maximum for `mclbytes`.

pfil_ipsec

Run packet filters on packets before encryption. The default is to do it after encryption.

pkt_cache=*X*

As `mbuf` and cluster combinations are freed after use, rather than return them to the internal pool for general consumption, up to *X* `mbufs` and clusters are cached per thread to allow quicker retrieval on the next allocation.

pkt_typed_mem=*object*

Allocate packet buffers from the specified typed memory object. For example:

```
io-pkt -ptcpip pkt_typed_mem=ram/dma
```

prefix=*/path*

The path to prepend to the traditional **/dev/socket**. The is useful when running multiple stacks (see the `-i` option). Clients can target a particular stack by using the *SOCK* environment variable. For example:

```
io-pkt -i1 -ptcpip prefix=/alt
SOCK=/alt ifconfig -a
```

recv_ctxt=*X*

Specify the size of the receive context buffer, in bytes. The default is 65536; the minimum is 2048.

reply_ctxt=*X*

Specify the number of buffer objects that `io-pkt` can send in reply to an application in one kernel operation. This setting does not limit the amount of data that can be included in the reply, it just determines how many operations are needed to reply with a given amount of data. The default is 90 objects; the minimum is 32.

reuseport_unicast

If using the SO_REUSEPORT socket option, received unicast UDP packets are delivered to all sockets bound to the port. The default is to deliver only multicast and broadcast to all sockets.

rx_prio=*X* or **rx_pulse_prio=*X***

The priority for receive threads to use (the default is 21). A driver-specific priority option (if supported by the driver) can override this priority.

smmu=0|1|off|on

Specify whether or not support for the system memory management unit (IOMMU/SMMU) manager is required:

- 0 or `off` — disable SMMU support. This is the default.
- 1 or `on` — SMMU support is required; `io-pkt` exits if it isn't available

If *value* isn't valid, `io-pkt` disables SMMU support and sends a message to `slogger2`.

For more information, see the *SMMUMAN User's Guide*.

so_txprio_enable

Enable the SO_TXPRIO socket option (see *getsockopt()* in the QNX Neutrino *C Library Reference*).

The SO_TXPRIO socket option sets the transmit queue priority on a socket. If you set this priority, then all traffic sent through the socket carries a packet tag of type PACKET_TAG_TXQ whose value is the priority value that you set with *setsockopt()*. If you don't start `io-pkt` with this option, and you then try to set SO_TXPRIO, *setsockopt()* fails and sets *errno* to EOPNOTSUPP.

The `so_txprio_enable` option restricts only the capability of the SO_TXPRIO socket option. It has no effect on **lsm-llmcast.so** or **lsm-avb.so**, which both can generate packets that carry PACKET_TAG_TXQ tags.

Setting the `so_txprio_enable` option negatively affects the performance of `io-pkt`. This is true even if application code doesn't call *setsockopt()* to set the value of SO_TXPRIO. You should enable this option only if you need to set the transmit queue priority.

To complete the priority transmit queue function, a network driver must extract the priority value from the packet's PACKET_TAG_TXQ tag, and then enqueue the packet on the appropriate transmit queue. If a network driver doesn't support extracting PACKET_TAG_TXQ, then it can treat a packet as an ordinary one, and setting the SO_TXPRIO option has no useful effect, other than slowing down the traffic because of the extra work. Therefore application developers should use this option only with a network driver that supports priority transmit queues.

The `io-pkt` stack code (excluding network drivers) doesn't interpret this priority value in any way. Specifically, `io-pkt` doesn't associate the transmit queue priority with IP_TOS or IPV6_TCLASS. If you want to associate a transmit queue priority value with IP_TOS or IPV6_TCLASS, you must properly set both IP_TOS (or IPV6_TCLASS) and SO_TXPRIO on the socket, so that traffic through it has the proper type of service and is transmitted at the proper priority by a network driver.

somaxconn=*X*

Specify the value of SOMAXCONN, the maximum length of the listen queue used to accept new TCP connections. The minimum is the value in **<sys/socket.h>**.

stackguard

Introduce a guard page between each thread's stack to aid in debugging “blown stack handling” panics. This will cause a SIGSEGV at the point of stack overrun rather than at the end of the operation.

If the value of the `stacksize` option isn't a multiple of the system page size, then this option increases the stack size until it is. A message is logged to `slogger2` in this case advising of the new size. This increase in stack size may change the issue being debugged.

stacksize=*X*

Specify the size of each thread's stack, in bytes. The default is 4096 (4 KB) in 32-bit architectures, and 8192 (8 KB) in 64-bit architectures.

strict_ts

(QNX Neutrino 7.0.1 or later) Use the `io-pkt` timer for timestamping BPF. This results in a best-case precision of 1 millisecond, but the timestamps are guaranteed to be monotonic. By default *ClockCycles()* is used to give timestamps that are precise down to 1 microsecond, at the risk of occasional nonmonotonic timestamps as the *ClockCycles()* clock is recalibrated against the `io-pkt` timer.

threads_incr=*X*

If the supply of functional connections is exhausted, increment their number by this amount, up to the value of `threads_max`. The default is 25.

The term "threads" in the TCP/IP `threads_*` options is a misnomer; it really refers to functional TCP/IP connections or blocking operations (*read()*, *write()*, *accept()*, etc.). It has nothing to do with the number of threads running in `io-pkt-*`.

threads_max=*X*

Specify the maximum number of functional TCP/IP connections that the stack can service simultaneously. The default is 200.

threads_min=*X*

Specify the minimum number of functional TCP/IP connections. The default is 15, and the minimum is 4.

timer_pulse_prio=*priority*

The priority to use for the timer pulse. The default is 21.

Description:

The `io-pkt` manager provides support for Internet domain sockets, Unix domain sockets, and dynamically loaded networking modules. It comes in the following stack variants:

io-pkt-v4-hc

IPv4 version of the stack that has full encryption and Wi-Fi capability built in and includes hardware-accelerated cryptography capability (Fast IPsec).

io-pkt-v6-hc

IPv6 version of the stack (includes IPv4 as part of v6) that has full encryption and Wi-Fi capability, also with hardware-accelerated cryptography.

In order to use SSL connections, you must have started `random` with the `-t` option.

After you've launched `io-pkt*`, you can use the `mount` command to start drivers or load additional modules such as **lsm-pf-v4.so** or **lsm-pf-v6.so**. If you want to pass options to the driver, use the `-o` option *before* the name of the shared object. For example:

```
mount -T io-pkt -o mac=12345678 devnp-abc100.so
```

- You can't use `umount` to unmount `io-pkt*` drivers. You might be able to detach the driver from the stack by using `ifconfig`'s `destroy` command (if the driver supports it).
- If `io-pkt` runs out of threads, it sends a message to `slogger2`, and anything that requires a thread blocks until one becomes available.
- The network drivers don't put entries into the **/dev** namespace, so a `waitfor` command for such an entry won't work properly in buildfiles or scripts. Use `if_up -p` instead; for example, `if_up -p en0`.
- If a TCP/IP packet is smaller than the minimum Ethernet packet size, the packet may be padded with random data, rather than zeroes.

The `io-pkt` manager supports TUN and TAP. To create the interfaces, use `ifconfig`:

```
ifconfig tun0 create
ifconfig tap0 create
```

For more information, see the NetBSD documentation:

- **http://netbsd.gw.com/cgi-bin/man-cgi?tun++NetBSD-current**
- **http://netbsd.gw.com/cgi-bin/man-cgi?tap++NetBSD-current**

Generic driver options

The stack processes the following generic driver options:

`name=`*`prefix`*

Override the default interface prefix used for network drivers. For example:

```
io-pkt-v4-hc -d abc100 name=en
```

starts the fictitious **devnp-abc100.so** driver with the "en" interface naming convention (`en`*XX*). You can also use this option to assign interface names based on (for example) functionality:

```
io-pkt-v4-hc -d abc100 pci=0,name=wan
```

`unit=`*`number`*

The interface number to use. If *number* is negative, it's ignored. By default, the interfaces are numbered starting at 0.

The stack also processes the following driver options for all USB drivers using the NetBSD-to-QNX conversion library to let you identify a particular USB device using information obtained from running `usb -v`:

did=*ID*

Device product ID.

vid=*ID*

Device vendor ID.

devno=*addr*

Device address, as reported by the `usb` utility.

busno=*num*

Host controller, as reported by the `usb` utility

For example:

```
io-pkt-v4-hc -d abc100 did=0x0020,vid=0x13b1,devno=1,busno=1
```

Examples:

Start the v6 variant of `io-pkt` using the fictitious **devnp-abc100.so** driver:

```
io-pkt-v6-hc -d /lib/dll/devnp-abc100.so \
  memrange=0x10064000,irq=0x80050024,mac=001122334455
ifconfig abc0 10.184
```

Chapter 3

pci_device_reset()

Reset a device

Synopsis:

```
#include <pci/pci.h>

pci_err_t pci_device_reset( pci_devhdl_t hdl,
                           pci_resetType_e resetType );
```

Arguments:

hdl

The handle of the device, obtained by calling *pci_device_attach()*.

resetType

The type of reset to initiate. The defined types are:

- pci_resetType_e_BUS
- pci_resetType_e_FUNCTION
- pci_resetType_e_SEGMENT

A reset type that's less than pci_resetType_e_BUS is considered to mean “no reset” (a no-op).

A reset type that's greater than pci_resetType_e_FUNCTION is considered to be a hardware-specific reset. With the exception of pci_resetType_e_SEGMENT, the behavior of hardware-specific resets, if supported, is defined by the hardware-dependent module. Refer to the release notes or the usage information in the hardware-dependent module applicable to your platform for details.

Library:

libpci

Use the `-l pci` option to `qcc` to link against this library.

Description:

The *pci_device_reset()* function initiates a reset of a specific device (function-level reset if the device is PCIe and supports FLR as determined by the device capabilities register or if the device is PCI and supports the Advanced Features capability and FLR as per the AF capabilities register) or of a PCI bus segment/PCIe link (secondary bus reset) if the device is a PCI-to-PCI bridge or PCIe Root Port that reports as a PCI-to-PCI bridge.

The *hdl* arguments identifies the specific device to be reset, or the bus or link on which the device resides.

Since this API can potentially impact multiple devices, you must be sure of what you're doing. During the time that the reset is occurring, configuration space accesses to the device(s) is halted, but there's currently no in-band mechanism for preventing access to device-specific registers obtained from a successful call to *pci_device_read_ba()* and *mmap()*'d by other driver software. It's your responsibility to coordinate with such software as required.

The primary types of reset are:

pci_resetType_e_BUS

Reset all devices on the bus or link on which the device identified by *hdl* resides, including any downstream buses or links for subordinate bridges.

pci_resetType_e_FUNCTION

Reset only the specific device identified by *hdl*.

pci_resetType_e_SEGMENT

This reset is similar to a pci_resetType_e_BUS reset with respect to the reconfiguration of devices below the root port. That is, all devices below the root port are returned to their D0 initialized state similar to the case of a pci_resetType_e_BUS reset issued on the downstream link of a root port, however the root port itself is also reconfigured to the D0 initialized state. The primary difference with this reset type is that it's the responsibility of the hardware module to actually reset the root port in a hardware-specific fashion. If the hardware module doesn't support this reset type, *pci_device_reset()* returns PCI_ERR_ENOTSUP.

You *must* be an owner of the device identified by *hdl* and the only remaining attacher.

Affected devices

For a function reset, only the device identified by *hdl* is reset. This includes a specific function of a multifunction device. You must be an owner of the device and the only remaining attacher (all other multiowned attachers, if any, must have detached). In addition, for bus/link resets, all devices that reside on the same bus/link as the device identified by *hdl* or that reside on a bus/link that's downstream of the bus/link on which the device identified by *hdl* resides are also reset. In this case, those devices must not have any attachments (i.e., no other software must have successfully called *pci_device_attach()* on these devices without having called *pci_device_detach()* prior to the reset operation). It's the responsibility of the application software to coordinate these operations for other affected device software, or the *pci_device_reset()* call will fail.

A reset of a specific device (i.e., a BDF) is supported by the PCI specification-defined Function Level Reset (FLR) mechanism. In order to initiate this type of reset, the device must either be a PCIe endpoint with bit 28 of the device capabilities register set to binary value 1 or have the Advanced Features (AF) capability with bit 1 of the AF capabilities register set to binary value 1, otherwise this reset is unsupported.

Other reset types are allowed, but these have no defined behavior within the PCI server, and so are passed directly to the hardware-dependent module. This allows for platform-specific reset processing

to be accommodated. It's the responsibility of the hardware-dependent module to document what additional reset types if any it supports and what the behaviors of those resets are.

Although the hardware-dependent module has complete control over the reset behavior, the PCI server ensures that configuration space accesses to the device being reset are halted, and after reset, that the device is configured into the D0-initialized state unless otherwise prevented from doing so by the hardware-dependent module. All of the discussions that follow regarding post-reset state and interrupts are applicable to hardware-dependent reset types unless the hardware-dependent module documents otherwise.

Post-reset driver state

On return from a successful *pci_device_reset()*, your *hdl* is still valid, as are any mappings to address spaces obtained with a successful call to *pci_device_read_ba()*, but all capabilities are disabled. In order to use the capabilities, you need to:

- reread them, using *pci_device_read_cap()*; you can reuse the same `pci_cap_t`
- reconfigure them (as required using the appropriate capability-specific APIs)
- reenable them by calling *pci_device_cfg_cap_enable()*

Device software for other affected devices (if any), must go through the entire initialization phase starting with *pci_device_attach()*, because they were required to detach in order for the reset to take place.

Regarding interrupts

If the device was configured to use the MSI or MSI-X capability, the IRQs associated with the MSI/MSI-X vectors are released when the capability is disabled. Since MSI/MSI-X vectors are allocated when the capability is enabled, and these could potentially change between the disabling and reenabling of these capabilities, driver software should call *InterruptDetach()* for previously assigned IRQs, reread them with *pci_device_read_irq()*, and then reattach them with *InterruptAttach()* or *InterruptAttachEvent()*.

If you aren't using MSI or MSI-X, you don't need to reread the pin-based IRQs originally returned because they don't change. That is, there's no need for the driver software to call *InterruptDetach()*, *pci_device_read_irq()*, and then *InterruptAttach()* or *InterruptAttachEvent()* after the reset if using pin-based (i.e., non-MSI or non-MSI-X) interrupts.

Returns:

If successful, a call to *pci_device_reset()* with a reset type of either pci_resetType_e_BUS or pci_resetType_e_FUNCTION returns PCI_ERR_OK, and all affected devices (as described above) have been reset and reconfigured to the D0-initialized state. Any driver software using those device must comprehend this condition and perform any required device-specific initialization that may have been lost as a result of the reset operation.

For reset types other than pci_resetType_e_BUS or pci_resetType_e_FUNCTION, the hardware-dependent module must document the post-reset state of affected devices.

If any error occurs, *pci_device_reset()* returns one of the following values, and you should consider all affected devices (as outlined above) to be in an unknown and unusable state. It this situation, it may be possible, and even desirable to reissue the reset command.

PCI_ERR_ENOTSUP

The reset type specified isn't supported by the device identified by *hdl*.

PCI_ERR_EINVAL

The *hdl* doesn't refer to a valid device that you attached to, or other parameters are otherwise invalid. The hardware-dependent module can also return this error.

PCI_ERR_ENODEV

The device identified by *hdl* doesn't exist. Note that this error can also be returned if a device that supports live removal is removed.

PCI_ERR_NOT_OWNER

You don't own the device identified by *hdl*.

PCI_ERR_ATTACH_SHARED

The device identified by *hdl* is currently attached to by more than just you.

PCI_ERR_ATTACH_OWNED

Devices on the same bus or link as the device identified by *hdl* or on a downstream bus or link are still attached.

Classification:

QNX Neutrino

Safety:	
Cancellation point	No
Interrupt handler	No
Signal handler	No
Thread	Yes

Chapter 4
PCIe allocate-once policy for device interrupts

The allocate-once policy for device interrupts eliminates the need for a device driver to call *InterruptAttach()* or *InterruptAttachEvent()* more than once during its lifetime, so that then the driver can drop the privileges required to make those API calls.

This functionality is disabled by default; to enable it, set the `ALLOCATE_ONCE_IRQ_POLICY` parameter in the `[nv_t19x]` section of a hardware configuration file. For example:

```
[nv_t19x]
# Enable the allocate-once IRQ policy.
# The parameter value must be a case-invariant 'true'
ALLOCATE_ONCE_IRQ_POLICY=true

# Disable the allocate-once IRQ Policy (the default behavior).
# The parameter value must be a case-invariant 'false'
# ALLOCATE_ONCE_IRQ_POLICY=false
```

For details on using a hardware configuration file, see ***$QNX_TARGET*/etc/system/config/pci/pci_hw-template.cfg**.

In order for this functionality to work reliably, note the following:

- Once IRQs have been successfully allocated to a device, those same IRQs continue to be assigned or reassigned to the device until one of the following conditions occurs:
  - The `pci-server` is restarted.
  - The EP driver requests a different interrupt type (e.g., INTPIN VS MSI).
  - The EP driver requests more interrupts than were originally assigned to it.
- If using message-based interrupts, drivers should use either the MSI or MSI-X capability exclusively and avoid alternating between the two. If the number of supported and/or driver-configured interrupts differs between these two capabilities, there's a risk that a release and reallocation will occur if the EP driver requests more interrupts than were originally assigned to it.
- Drivers should not mix interrupt types.

Chapter 5
ptpd, ptpd-avb

Precision Time Protocol daemon

Syntax:

```
ptpd [-?BCcDdEGghjKLPQStUWx] [-A] [-a number,number] [-b name]
     [ -e arg] [-F number:number] [-f file] [-I group] [-i number]
     [-J number] [-k number] [-l number,number] [-M number]
     [-m number] [-N number] [-n number] [-O number] [-o number]
     [-p number] [-q number] [-R file] [-r number] [-s number]
     [-T ttl][-u address] [-V number] [-v number] [-w number]
     [-X number] [-Y number] [-y number] [-Z]

ptpd-avb [-?BCcDdEGghjKLQPStUWx] [-A] [-a number,number] [-b name]
         [ -e arg] [-F number:number] [-f file] [-I group] [-i number]
         [-J number] [-k number] [-l number,number] [-M number]
         [-m number] [-N number] [-n number] [-O number] [-o number]
         [-p number] [-q number] [-R file] [-r number] [-s number]
         [-T ttl] [-u address] [-V number] [-v number] [-w number]
         [-X number] [-Y number] [-y number] [-Z]
```

Runs on:

QNX Neutrino

Options:

-?

Display a short help text.

-A

When set, peer delay (or path delay) requests are ignored when running as a slave.

-a *number*,*number*

Specify clock servo P and I attenuations.

-B

Enable debugging if it has been previously compiled in.

-b *name*

Bind PTP to the network interface *name*.

-C

Run in command-line mode, and display statistics and logs.

-c

Run in command-line (nondaemon) mode.

-D

Display stats in **.csv** format.

-d

Display stats.

-E

Enable AI/AP optimization.

-e ***arg***

The *arg* can be one of the following:

- `noHWClkAdj` — don't update the HW PTP clock.
- `noSysClkAdj` — don't update the system clock.
- `noSetPtpTimeFromSys` — during initialization time and state reset, don't reset the PTP clock using the value of the system clock. This keeps the PTP clock completely free from the effects of the system clock.

-F ***ClockId*:*PortId***

As a slave, force Master Clock Identity and ignore announce packets. `0000000000000000:0` is a wildcard to use the Clock Identity from the first received sync packet.

As a master, don't run the Best Master Clock Algorithm (BMCA); immediately go to master status. The clock and port IDs are ignored in master mode.

-f ***file***

Send output to *file*.

-G

Run as master with connection to NTP.

-g

Run as slave only.

-h

Run in End-to-End mode.

-I ***group***

Specify the multicast group for PTP_EXPERIMENTAL mode.

-i *number*

Specify the PTP domain number.

-J *number*

Specify the maximum offset in ns before stepping the clock.

-j

Turn off IGMP refresh messages.

-K

Enable *devctl()* support. If you specify this option, `ptpd` and `ptpd-avb` register **/dev/ptpd** (DEFAULT_PTPD_PATH, defined in **<dcmd_ptpd.h>**) in the path namespace; use the DCMD_PTPD_* commands with a file descriptor from opening this path. For more information, see *Devctl and Ioctl Commands*.

-k *number*

Specify the fixed number of threads to be created for the resource manager. If you specify this option, `-K` is automatically set as well.

-L

Enable the running of multiple `ptpd` daemons.

-l *number*,*number*

("el") Specify inbound and outbound latencies, in nsec.

-M *number*

Don't accept delay values of more than *number* nanoseconds.

-m *number*

Specify the maximum number of foreign master records.

-N *number*

Specify the announce receipt timeout.

-n *number*

Specify the announce interval in 2^*number* sec.

-O *number*

("Oh") Don't reset the clock if the offset is more than *number* nanoseconds.

-o *number*

Specify the current UTC offset.

-P

Display packets received for debugging purposes.

-p ***number***

Specify the priority1 attribute.

-Q

Don't handle signalling messages.

-q ***number***

Specify the priority2 attribute.

-R ***file***

Record a quality *file*.

-r ***number***

Specify the system clock accuracy.

-S

Enable logging to `slogger2`.

-s ***number***

Specify the system clock class.

-T ***ttl***

Set multicast TTL for packets. Defaults to 1.

-t

Don't adjust the system clock.

-U

Enable hybrid mode, which uses both unicast and multicast; requires PTP_EXPERIMENTAL.

-u ***address***

Also send unicast to *address*.

-V ***number***

Limit displaying statistics by setting the seconds between log messages.

-v ***number***

Specify the system clock allen variance.

-W

Run as master without NTP.

-w ***number***

Specify one-way delay filter stiffness.

-X ***number***

Specify the correction (ns) to add to the offset when stepping the clock.

-x

Don't reset the clock if it's off by more than one second.

-Y ***number***

Set an initial delay request value.

-y ***number***

Specify the sync interval in 2^*number* sec.

-Z

When set, path trace type-length-values (TLVs) are added to the announce messages.

Description:

The `ptpd` daemon implements the Precision Time Protocol (PTP) Version 2 as defined by the IEEE 1588-2008 standard. PTP was developed to provide very precise time coordination of LAN-connected computers.

The `ptpd-avb` and `ptpd` daemons are similar, but `ptpd` time update datagrams are encapsulated in an IP packet on top of Ethernet (IEEE-1588v2 protocol specification), whereas `ptpd-avb` time update datagrams are directly on Ethernet (IEEE-802.1AS protocol specification).

PTPd is a complete implementation of the IEEE 1588 v2 specification for a standard (ordinary) clock. PTPd has been tested with and is known to work properly with other IEEE 1588 implementations. The source code for PTPd is freely available under a BSD-style license. Thanks to contributions from users, PTPd is becoming an increasingly portable, interoperable, and stable IEEE 1588 implementation.

For more information, see **http://ptpd.sourceforge.net/**.

Contributing authors:

Gael Mace (**gael_mace@users.sourceforge.net**), Alexandre Van Kempen, Steven Kreuzer (**skreuzer@freebsd.org**), and George Neville-Neil (**gnn@freebsd.org**)

Chapter 6
shutdown

Shut down and reboot the system (QNX Neutrino)

You must be **root** to run this utility.

Syntax:

```
shutdown [-bcfkqvw] [-n nodename] [-S type]
```

Runs on:

QNX Neutrino

Options:

-b

Shut down but don't reboot. You can't use this option with `-n` *nodename*.

-c

Kill processes in the reverse of the order in which they were created (i.e., kill the newest one first).

-f

Shut down fast. Reduce the amount of time between sending a SIGTERM signal and a sending a SIGKILL to processes that catch SIGTERM.

-k

Shut down, and activate the kill switch. If there isn't a kill switch, then perform a system shutdown. This option might not be supported on all hardware.

-n *nodename*

Shut down the specified node (default is current node).

-q

Be quiet.

-S *type*

The type of shutdown, which must be one of:

- `system` — shut down the system.
- `reboot` — reboot the system.

The default is `reboot`.

-v

Be verbose.

-w

Do a warm reboot, if possible.

Description:

The `shutdown` utility performs an orderly system shutdown. In general terms, `shutdown` does the following for each process listed under **/proc**:

1. It sends a SIGTERM signal to the process if it isn't a critical process.
2. (QNX Neutrino 6.6 or later) It sends a SIGCONT signal, in case the process was stopped.
3. It sends a SIGPWR signal to the process.
4. It waits for a period of time. If the *SIGKILL_TIMEOUT* environment variable is defined, its value is used as the number of milliseconds to wait. Otherwise, the time depends on the class of the application (reduced if you specify the `-f` option).
5. It sends a SIGKILL signal to the process if it still exists and it isn't a critical process.

It then reboots the system (unless you specified the `-b` option).

The interval between the SIGTERM and SIGKILL signals allows processes that have elected to catch the SIGTERM signal to perform any cleanup they need to do before the system is rebooted. The SIGCONT allows a stopped process to be terminated by the queued SIGTERM or the subsequent SIGKILL.

This utility tries to shut down the processes in a reasonable—but generic—order. You might want to create your own utility that shuts down your system in a particular way.

Files:

/var/log/wtmp

If this file already exists, `shutdown` adds an entry to it before shutting down or rebooting the system.

The `shutdown` utility doesn't create **/var/log/wtmp** if it doesn't already exist. This file can quickly become very big, which isn't good on an embedded system with limited resources.

Environment variables:

SIGKILL_TIMEOUT

If this environment variable is defined, its value is used as the number of milliseconds to wait before sending a SIGKILL signal to processes that haven't yet terminated.