## NAME

**nvme** - NVM Express core driver

## SYNOPSIS

To compile this driver into your kernel, place the following line in your kernel configuration file:

```
device nvme
```

Or, to load the driver as a module at boot, place the following line in loader.conf(5):

```
nvme_load="YES"
```

Most users will also want to enable nvd(4) or nda(4) to expose NVM Express namespaces as disk devices which can be partitioned. Note that in NVM Express terms, a namespace is roughly equivalent to a SCSI LUN.

## DESCRIPTION

The **nvme** driver provides support for NVM Express (NVMe) controllers, such as:

- Hardware initialization
- Per-CPU IO queue pairs
- API for registering NVMe namespace consumers such as nvd(4) or nda(4)
- API for submitting NVM commands to namespaces
- Ioctls for controller and namespace configuration and management

The **nvme** driver creates controller device nodes in the format */dev/nvmeX* and namespace device nodes in the format */dev/nvmeXnsY*. Note that the NVM Express specification starts numbering namespaces at 1, not 0, and this driver follows that convention.

## CONFIGURATION

By default, **nvme** will create an I/O queue pair for each CPU, provided enough MSI-X vectors and NVMe queue pairs can be allocated. If not enough vectors or queue pairs are available, nvme(4) will use a smaller number of queue pairs and assign multiple CPUs per queue pair.

To force a single I/O queue pair shared by all CPUs, set the following tunable value in loader.conf(5):

hw.nvme.per_cpu_io_queues=0

To assign more than one CPU per I/O queue pair, thereby reducing the number of MSI-X vectors consumed by the device, set the following tunable value in loader.conf(5):

hw.nvme.min_cpus_per_ioq=X

To force legacy interrupts for all **nvme** driver instances, set the following tunable value in loader.conf(5):

hw.nvme.force_intx=1

Note that use of INTx implies disabling of per-CPU I/O queue pairs.

To control maximum amount of system RAM in bytes to use as Host Memory Buffer for capable devices, set the following tunable:

hw.nvme.hmb_max

The default value is 5% of physical memory size per device.

The nvd(4) driver is used to provide a disk driver to the system by default. The nda(4) driver can also be used instead. The nvd(4) driver performs better with smaller transactions and few TRIM commands. It sends all commands directly to the drive immediately. The nda(4) driver performs better with larger transactions and also collapses TRIM commands giving better performance. It can queue commands to the drive; combine BIO_DELETE commands into a single trip; and use the CAM I/O scheduler to bias one type of operation over another. To select the nda(4) driver, set the following tunable value in loader.conf(5):

hw.nvme.use_nvd=0

This value may also be set in the kernel config file with

**options NVME_USE_NVD=0**

When there is an error, **nvme** prints only the most relevant information about the command by default. To enable dumping of all information about the command, set the following tunable value in loader.conf(5):

hw.nvme.verbose_cmd_dump=1

Prior versions of the driver reset the card twice on boot. This proved to be unnecessary and inefficient, so the driver now resets drive controller only once. The old behavior may be restored in the kernel config file with

**options NVME_2X_RESET**

## SYSCTL VARIABLES

The following controller-level sysctls are currently implemented:

*dev.nvme.0.num_cpus_per_ioq*
(R) Number of CPUs associated with each I/O queue pair.

*dev.nvme.0.int_coal_time*
(R/W) Interrupt coalescing timer period in microseconds. Set to 0 to disable.

*dev.nvme.0.int_coal_threshold*
(R/W) Interrupt coalescing threshold in number of command completions. Set to 0 to disable.

The following queue pair-level sysctls are currently implemented. Admin queue sysctls take the format of dev.nvme.0.adminq and I/O queue sysctls take the format of dev.nvme.0.ioq0.

*dev.nvme.0.ioq0.num_entries*
(R) Number of entries in this queue pair’s command and completion queue.

*dev.nvme.0.ioq0.num_tr*
(R) Number of nvme_tracker structures currently allocated for this queue pair.

*dev.nvme.0.ioq0.num_prp_list*
(R) Number of nvme_prp_list structures currently allocated for this queue pair.

*dev.nvme.0.ioq0.sq_head*
(R) Current location of the submission queue head pointer as observed by the driver. The head pointer is incremented by the controller as it takes commands off of the submission queue.

*dev.nvme.0.ioq0.sq_tail*
(R) Current location of the submission queue tail pointer as observed by the driver. The driver increments the tail pointer after writing a command into the submission queue to signal that a new command is ready to be processed.

*dev.nvme.0.ioq0.cq_head*

(R) Current location of the completion queue head pointer as observed by the driver. The driver increments the head pointer after finishing with a completion entry that was posted by the controller.

*dev.nvme.0.ioq0.num_cmds*
(R) Number of commands that have been submitted on this queue pair.

*dev.nvme.0.ioq0.dump_debug*
(W) Writing 1 to this sysctl will dump the full contents of the submission and completion queues to the console.

In addition to the typical pci attachment, the **nvme** driver supports attaching to a ahci(4) device. Intel's Rapid Storage Technology (RST) hides the nvme device behind the AHCI device due to limitations in Windows. However, this effectively hides it from the FreeBSD kernel. To work around this limitation, FreeBSD detects that the AHCI device supports RST and when it is enabled. See ahci(4) for more details.

## SEE ALSO

nda(4), nvd(4), pci(4), nvmecontrol(8), disk(9)

## HISTORY

The **nvme** driver first appeared in FreeBSD 9.2.

## AUTHORS

The **nvme** driver was developed by Intel and originally written by Jim Harris <*jimharris@FreeBSD.org*>, with contributions from Joe Golio at EMC.

This man page was written by Jim Harris <*jimharris@FreeBSD.org*>.