NAME

CAM - Common Access Method Storage subsystem

SYNOPSIS

device scbus
device ada
device cd
device ch
device da
device pass
device pt
device sa
options CAMDEBUG
options CAM_DEBUG_BUS=-1
options CAM_DEBUG_TARGET=-1
options CAM_DEBUG_LUN=-1
options
CAM_DEBUG_COMPILE=CAM_DEBUG_INFO|CAM_DEBUG_CDB|CAM_DEBUG_PROBE
options CAM_DEBUG_FLAGS=CAM_DEBUG_INFO|CAM_DEBUG_CDB
options CAM_MAX_HIGHPOWER=4
options SCSI_NO_SENSE_STRINGS
options SCSI_NO_OP_STRINGS
options SCSI_DELAY=8000

DESCRIPTION

The **CAM** subsystem provides a uniform and modular system for the implementation of drivers to control various SCSI, ATA, NVMe, and MMC / SD devices, and to utilize different SCSI, ATA, NVMe, and MMC / SD host adapters through host adapter drivers. When the system probes buses, it attaches any devices it finds to the appropriate drivers. The pass(4) driver, if it is configured in the kernel, will attach to all devices.

KERNEL CONFIGURATION

There are a number of generic kernel configuration options for the **CAM** subsystem:

CAM_BOOT_DELAY — Additional time to wait after the static parts of the kernel have run to allow for discovery of additional devices which may take time to connect, such as USB attached storage.

CAM_IOSCHED_DYNAMIC — Enable dynamic decisions in the I/O scheduler based on hints and the

current performance of the storage devices.

CAM_IO_STATS Enable collection of statistics for periph devices.

CAM_TEST_FAILURE Enable ability to simulate I/O failures.

CAMDEBUG This option compiles in all the **CAM** debugging printf code. This will not actually cause any debugging information to be printed out when included by itself. See below for details.

CAM_MAX_HIGHPOWER=4
This sets the maximum allowable number of concurrent "high power" commands. A "high power" command is a command that takes more electrical power than most to complete. An example of this is the SCSI START UNIT command. Starting a disk often takes significantly more electrical power than normal operation. This option allows the user to specify how many concurrent high power commands may be outstanding without overloading the power supply on his computer.

SCSI_NO_SENSE_STRINGS This eliminates text descriptions of each SCSI Additional Sense Code and Additional Sense Code Qualifier pair. Since this is a fairly large text database, eliminating it reduces the size of the kernel somewhat. This is primarily necessary for boot floppies and other low disk space or low memory space environments. In most cases, though, this should be enabled, since it speeds the interpretation of SCSI error messages. Do not let the "kernel bloat" zealots get to you -- leave the sense descriptions in your kernel!

SCSI_NO_OP_STRINGS This disables text descriptions of each SCSI opcode. This option, like the sense string option above, is primarily useful for environments like a boot floppy where kernel size is critical. Enabling this option for normal use is not recommended, since it slows debugging of SCSI problems.

SCSI_DELAY=8000 This is the SCSI "bus settle delay." In **CAM**, it is specified in *milliseconds*, not seconds like the old SCSI layer used to do. When the kernel boots, it sends a bus reset to each SCSI bus to tell each device to reset itself to a default set of transfer negotiations and other settings. Most SCSI devices need some amount of time to recover from a bus reset. Newer disks may need as little as 100ms, while old, slow devices may need much longer. If the SCSI_DELAY is not specified, it defaults

to 2 seconds. The minimum allowable value for SCSI_DELAY is "100", or 100ms. One special case is that if the SCSI_DELAY is set to 0, that will be taken to mean the "lowest possible value." In that case, the SCSI_DELAY will be reset to 100ms.

All devices and buses support dynamic allocation so that an upper number of devices and controllers does not need to be configured; **device da** will suffice for any number of disk drivers.

The devices are either *wired* so they appear as a particular device unit or *counted* so that they appear as the next available unused unit.

Units are wired down by setting kernel environment hints. This is usually done either interactively from the loader(8), or automatically via the */boot/device.hints* file. The basic syntax is:

```
hint.device.unit.property="value"
```

Individual **CAM** bus numbers can be wired down to specific controllers with a config line similar to the following:

```
hint.scbus.0.at="ahd1"
```

This assigns **CAM** bus number 0 to the *ahd1* driver instance. For controllers supporting more than one bus, a particular bus can be assigned as follows:

```
hint.scbus.0.at="ahc1"
hint.scbus.0.bus="1"
```

This assigns **CAM** bus 0 to the bus 1 instance on *ahc1*. Peripheral drivers can be wired to a specific bus, target, and lun as so:

```
hint.da.0.at="scbus0"
hint.da.0.target="0"
hint.da.0.unit="0"
```

This assigns *da0* to target 0, unit (lun) 0 of scbus 0. Omitting the target or unit hints will instruct **CAM** to treat them as wildcards and use the first respective counted instances. These examples can be combined together to allow a peripheral device to be wired to any particular controller, bus, target, and/or unit instance.

This also works with nvme(4) drives as well.

```
hint.nvme.4.at="pci7:0:0"
hint.scbus.10.at="nvme4"
hint.nda.10.at="scbus10"
hint.nda.10.target="1"
hint.nda.10.unit="12"
hint.nda.11.at="scbus10"
hint.nda.11.target="1"
hint.nda.11.unit="2"
```

This assigns the NVMe card living at PCI bus 7 slot 0 function 1 to scbus 10. The target for nda(4) devices is always 1. The unit is the namespace identifier from the drive. The namespace id 1 is exported as *nda10* and namespace id 2 is exported as *nda11*.

For devices that provide a serial number, units may be wired to that serial number without regard where the drive is attached:

```
hint.nda.3.sn="CY0AN07101120B12P"
hint.da.44.sn="143282400011"
hint.ada.2.sn="A065D591"
```

wires *nda3*, *da44*, and *ada2* to drives with the specified serial numbers. One need not specify an *at* line when serial numbers are used.

ADAPTERS

The system allows common device drivers to work through many different types of adapters. The adapters take requests from the upper layers and do all IO between the SCSI, ATA, NVMe, or MMC / SD bus and the system. The maximum size of a transfer is governed by the adapter. Most adapters can transfer 64KB in a single operation, however many can transfer larger amounts.

TARGET MODE

Some adapters support *target mode* in which the system is capable of operating as a device, responding to operations initiated by another system. Target mode is supported for some adapters, but is not yet complete for this version of the **CAM** SCSI subsystem.

ARCHITECTURE

The **CAM** subsystem glues together the upper layers of the system to the storage devices. PERIPH devices accept storage requests from GEOM and other upper layers of the system and translates them into protocol requests. XPT (transport) dispatches these protocol requests to a SIM driver. A SIM driver takes protocol requests and translates them into hardware commands the host adapter understands to transfer the protocol requests, and data (if any) to the storage device. The CCB transports these requests around as messages.

CAM

The Common Access Method was a standard defined in the 1990s to talk to disk drives. FreeBSD is one of the few operating systems to fully implement this model. The interface between different parts of CAM is the CCB (or CAM Control Block). Each CCB has a standard header, which contains the type of request and dispatch information, and a command specific portion. A CAM Periph generates requests. The XPT layer dispatches these requests to the appropriate SIM. Some CCBs are sent directly to the SIM for immediate processing, while others are queued and complete when the I/O has finished. A SIM takes CCBs and translates them into hardware specific commands to push the SCSI CDB or other protocol control block to the peripheral, along with setting up the DMA for the associated data.

Periph Devices

A periph driver knows how to translate standard requests into protocol messages that a SIM can deliver to hardware. These requests can come from any upper layer source, but primarily come in via GEOM as a bio request. They can also come in directly from character device requests for tapes and pass through commands.

Disk devices, or direct access (da) in CAM, are one type of peripheral. These devices present themselves to the kernel a device ending in "da". Each protocol has a unique device name:

da(4)
: SCSI or SAS device, or devices that accept SCSI CDBs for I/O.

ada(4)
: ATA or SATA device

nda(4)
: NVME device

sdda(4)
: An SD or MMC block storage device.

Tape devices are called serial access (sa(4)) in CAM. They interface to the system via a character device and provide ioctl(2) control for tape drives.

The pass(4) device will pass through CCB requests from userland to the SIM directly. The device is used to send commands other than read, write, trim or flush to a device. The camcontrol(8) command uses this device.

XPT drivers

The transport driver connects the periph to the SIM. It is not configured separately. It is also

responsible for device discovery for those SIM drivers that do not enumerate themselves.

SIM driver
SIM used to stand for SCSI Interface Module. Now it is just SIM because it understands protocols other than SCSI. There are two types of SIM drivers: virtual and physical. Physical SIMs are typically called host bus adapters (HBA), but not universally. Virtual SIM drivers are for communicating with virtual machine hosts.

FILES

see other **CAM** device entries.

DIAGNOSTICS

An XPT_DEBUG CCB can be used to enable various amounts of tracing information on any specific bus/device from the list of options compiled into the kernel. There are currently seven debugging flags that may be compiled in and used:

CAM_DEBUG_INFO	This flag enables general informational printfs for the device or devices in question.
CAM_DEBUG_TRACE	This flag enables function-level command flow tracing i.e., kernel printfs will happen at the entrance and exit of various functions.
CAM_DEBUG_SUBTRACE	This flag enables debugging output internal to various functions.
CAM_DEBUG_CDB	This flag will cause the kernel to print out all ATA and SCSI commands sent to a particular device or devices.
CAM_DEBUG_XPT	This flag will enable command scheduler tracing.
CAM_DEBUG_PERIPH	This flag will enable peripheral drivers messages.
CAM_DEBUG_PROBE	This flag will enable devices probe process tracing.

Some of these flags, most notably CAM_DEBUG_TRACE and CAM_DEBUG_SUBTRACE, will produce kernel printfs in EXTREME numbers.

Users can enable debugging from their kernel config file, by using the following kernel config options:

CAMDEBUG	This builds into the kernel all possible **CAM** debugging.

CAM_DEBUG_COMPILE This specifies support for which debugging flags described above should be built into the kernel. Flags may be ORed together if the user wishes to see printfs for multiple debugging levels.

CAM_DEBUG_FLAGS This sets the various debugging flags from a kernel config file.

CAM_DEBUG_BUS Specify a bus to debug. To debug all buses, set this to -1.

CAM_DEBUG_TARGET Specify a target to debug. To debug all targets, set this to -1.

CAM_DEBUG_LUN Specify a lun to debug. To debug all luns, set this to -1.

Users may also enable debugging on the fly by using the camcontrol(8) utility, if wanted options built into the kernel. See camcontrol(8) for details.

SEE ALSO

Commands:

camcontrol(8), camdd(8)

Libraries:

cam(3)

Periph Drivers:

ada(4), da(4), nda(4), pass(4), sa(4)

SIM Devices:

aac(4), aacraid(4), ahc(4), ahci(4), ata(4), aw_mmc(4), ciss(4), hv_storvsc(4), isci(4), iscsi(4), isp(4), mpr(4), mps(4), mpt(4), mrsas(4), mvs(4), nvme(4), pms(4), pvscsi(4), sdhci(4), smartpqi(4), sym(4), tws(4), umass(4), virtio_scsi(4)

Deprecated or Poorly Supported SIM Devices:

ahd(4), amr(4), arcmsr(4), esp(4), hpt27xx(4), hptiop(4), hptmv(4), hptnr(4), iir(4) mfi(4), sbp(4), twa(4)

HISTORY

The **CAM** SCSI subsystem first appeared in FreeBSD 3.0. The **CAM** ATA support was added in FreeBSD 8.0.

AUTHORS

The **CAM** SCSI subsystem was written by Justin Gibbs and Kenneth Merry. The **CAM** ATA support

was added by Alexander Motin <*mav@FreeBSD.org*>. The **CAM** NVMe support was added by Warner Losh <*imp@FreeBSD.org*>.