NAME

devstat, **devstat_end_transaction**, **devstat_end_transaction_bio**, **devstat_end_transaction_bio_bt**, **devstat_new_entry**, **devstat_remove_entry**, **devstat_start_transaction**, **devstat_start_transaction_bio** - kernel interface for keeping device statistics

SYNOPSIS

#include <sys/devicestat.h>

*struct devstat **
devstat_new_entry(*const void *dev_name*, *int unit_number*, *uint32_t block_size*, *devstat_support_flags flags*, *devstat_type_flags device_type*, *devstat_priority priority*);

void
devstat_remove_entry(*struct devstat *ds*);

void
devstat_start_transaction(*struct devstat *ds*, *const struct bintime *now*);

void
devstat_start_transaction_bio(*struct devstat *ds*, *struct bio *bp*);

void
devstat_end_transaction(*struct devstat *ds*, *uint32_t bytes*, *devstat_tag_type tag_type*, *devstat_trans_flags flags*, *const struct bintime *now*, *const struct bintime *then*);

void
devstat_end_transaction_bio(*struct devstat *ds*, *const struct bio *bp*);

void
devstat_end_transaction_bio_bt(*struct devstat *ds*, *const struct bio *bp*, *const struct bintime *now*);

DESCRIPTION

The devstat subsystem is an interface for recording device statistics, as its name implies. The idea is to keep reasonably detailed statistics while utilizing a minimum amount of CPU time to record them. Thus, no statistical calculations are actually performed in the kernel portion of the **devstat** code. Instead, that is left for user programs to handle.

The historical and antiquated **devstat** model assumed a single active IO operation per device, which is not accurate for most disk-like drivers in the 2000s and beyond. New consumers of the interface should almost certainly use only the "bio" variants of the start and end transacation routines.

devstat_new_entry() allocates and initializes *devstat* structure and returns a pointer to it. **devstat_new_entry**() takes several arguments:

dev_name The device name, e.g., da, cd, sa.

unit_number
Device unit number.

block_size Block size of the device, if supported. If the device does not support a block size, or if the blocksize is unknown at the time the device is added to the **devstat** list, it should be set to 0.

flags Flags indicating operations supported or not supported by the device. See below for details.

device_type The device type. This is broken into three sections: base device type (e.g., direct access, CDROM, sequential access), interface type (IDE, SCSI or other) and a pass-through flag to indicate pas-through devices. See below for a complete list of types.

priority The device priority. The priority is used to determine how devices are sorted within **devstat**’s list of devices. Devices are sorted first by priority (highest to lowest), and then by attach order. See below for a complete list of available priorities.

devstat_remove_entry() removes a device from the **devstat** subsystem. It takes the devstat structure for the device in question as an argument. The **devstat** generation number is incremented and the number of devices is decremented.

devstat_start_transaction() registers the start of a transaction with the **devstat** subsystem. Optionally, if the caller already has a **binuptime**() value available, it may be passed in **now*. Usually the caller can just pass NULL for *now*, and the routine will gather the current **binuptime**() itself. The busy count is incremented with each transaction start. When a device goes from idle to busy, the system uptime is recorded in the *busy_from* field of the *devstat* structure.

devstat_start_transaction_bio() records the **binuptime**() in the provided bio’s *bio_t0* and then invokes **devstat_start_transaction**().

devstat_end_transaction() registers the end of a transaction with the **devstat** subsystem. It takes six arguments:

ds The *devstat* structure for the device in question.

bytes The number of bytes transferred in this transaction.

tag_type Transaction tag type. See below for tag types.

flags Transaction flags indicating whether the transaction was a read, write, or whether no data was transferred.

now The **binuptime**() at the end of the transaction, or NULL.

then The **binuptime**() at the beginning of the transaction, or NULL.

If *now* is NULL, it collects the current time from **binuptime**(). If *then* is NULL, the operation is not tracked in the *devstat duration* table.

devstat_end_transaction_bio() is a thin wrapper for **devstat_end_transaction_bio_bt**() with a NULL *now* parameter.

devstat_end_transaction_bio_bt() is a wrapper for **devstat_end_transaction**() which pulls all needed information from a *struct bio* prepared by **devstat_start_transaction_bio**(). The bio must be ready for **biodone**() (i.e., *bio_bcount* and *bio_resid* must be correctly initialized).

The *devstat* structure is composed of the following fields:

sequence0,

sequence1 An implementation detail used to gather consistent snapshots of device statistics.

start_count Number of operations started.

end_count Number of operations completed. The "busy_count" can be calculated by subtracting *end_count* from *start_count*. (*sequence0* and *sequence1* are used to get a consistent snapshot.) This is the current number of outstanding transactions for the device. This should never go below zero, and on an idle device it should be zero. If either one of these conditions is not true, it indicates a problem.

There should be one and only one transaction start event and one transaction end event for each transaction.

dev_links Each *devstat* structure is placed in a linked list when it is registered. The *dev_links* field contains a pointer to the next entry in the list of *devstat* structures.

device_number
The device number is a unique identifier for each device. The device number is incremented for each new device that is registered. The device number is currently only a 32-bit integer, but it could be enlarged if someone has a system with more than four billion device arrival events.

device_name
The device name is a text string given by the registering driver to identify itself. (e.g., "da", "cd", "sa", etc.)

unit_number
The unit number identifies the particular instance of the peripheral driver in question.

bytes[4]
This array contains the number of bytes that have been read (index DEVSTAT_READ), written (index DEVSTAT_WRITE), freed or erased (index DEVSTAT_FREE), or other (index DEVSTAT_NO_DATA). All values are unsigned 64-bit integers.

operations[4]
This array contains the number of operations of a given type that have been performed. The indices are identical to those for *bytes* above. DEVSTAT_NO_DATA or "other" represents the number of transactions to the device which are neither reads, writes, nor frees. For instance, SCSI drivers often send a test unit ready command to SCSI devices. The test unit ready command does not read or write any data. It merely causes the device to return its status.

duration[4]
This array contains the total bintime corresponding to completed operations of a given type. The indices are identical to those for *bytes* above. (Operations that complete using the historical **devstat_end_transaction**() API and do not provide a non-NULL *then* are not accounted for.)

busy_time
This is the amount of time that the device busy count has been greater than zero. This is only updated when the busy count returns to zero.

creation_time
This is the time, as reported by **getmicrotime**() that the device was registered.

block_size
This is the block size of the device, if the device has a block size.

tag_types
This is an array of counters to record the number of various tag types that are sent to a device. See below for a list of tag types.

busy_from
If the device is not busy, this was the time that a transaction last completed. If the device is busy, this the most recent of either the time that the device became busy, or

the time that the last transaction completed.

flags   These flags indicate which statistics measurements are supported by a particular device. These flags are primarily intended to serve as an aid to userland programs that decipher the statistics.

device_type   This is the device type. It consists of three parts: the device type (e.g., direct access, CDROM, sequential access, etc.), the interface (IDE, SCSI or other) and whether or not the device in question is a pass-through driver. See below for a complete list of device types.

priority   This is the priority. This is the first parameter used to determine where to insert a device in the **devstat** list. The second parameter is attach order. See below for a list of available priorities.

id   Identification for GEOM nodes.

Each device is given a device type. Pass-through devices have the same underlying device type and interface as the device they provide an interface for, but they also have the pass-through flag set. The base device types are identical to the SCSI device type numbers, so with SCSI peripherals, the device type returned from an inquiry is usually ORed with the SCSI interface type and the pass-through flag if appropriate. The device type flags are as follows:

```
typedef enum {
        DEVSTAT_TYPE_DIRECT = 0x000,
        DEVSTAT_TYPE_SEQUENTIAL     = 0x001,
        DEVSTAT_TYPE_PRINTER        = 0x002,
        DEVSTAT_TYPE_PROCESSOR      = 0x003,
        DEVSTAT_TYPE_WORM   = 0x004,
        DEVSTAT_TYPE_CDROM = 0x005,
        DEVSTAT_TYPE_SCANNER        = 0x006,
        DEVSTAT_TYPE_OPTICAL        = 0x007,
        DEVSTAT_TYPE_CHANGER        = 0x008,
        DEVSTAT_TYPE_COMM   = 0x009,
        DEVSTAT_TYPE_ASC0   = 0x00a,
        DEVSTAT_TYPE_ASC1   = 0x00b,
        DEVSTAT_TYPE_STORARRAY      = 0x00c,
        DEVSTAT_TYPE_ENCLOSURE      = 0x00d,
        DEVSTAT_TYPE_FLOPPY = 0x00e,
        DEVSTAT_TYPE_MASK   = 0x00f,
        DEVSTAT_TYPE_IF_SCSI  = 0x010,
        DEVSTAT_TYPE_IF_IDE   = 0x020,
        DEVSTAT_TYPE_IF_OTHER           = 0x030,
        DEVSTAT_TYPE_IF_MASK            = 0x0f0,
        DEVSTAT_TYPE_PASS     = 0x100
} devstat_type_flags;
```

Devices have a priority associated with them, which controls roughly where they are placed in the **devstat** list. The priorities are as follows:

```
      typedef enum {
                DEVSTAT_PRIORITY_MIN            = 0x000,
                DEVSTAT_PRIORITY_OTHER          = 0x020,
                DEVSTAT_PRIORITY_PASS           = 0x030,
                DEVSTAT_PRIORITY_FD   = 0x040,
                DEVSTAT_PRIORITY_WFD            = 0x050,
                DEVSTAT_PRIORITY_TAPE           = 0x060,
                DEVSTAT_PRIORITY_CD   = 0x090,
                DEVSTAT_PRIORITY_DISK           = 0x110,
                DEVSTAT_PRIORITY_ARRAY          = 0x120,
                DEVSTAT_PRIORITY_MAX            = 0xfff
      } devstat_priority;
```

Each device has associated with it flags to indicate what operations are supported or not supported. The *devstat_support_flags* values are as follows:

DEVSTAT_ALL_SUPPORTED Every statistic type is supported by the device.

DEVSTAT_NO_BLOCKSIZE This device does not have a blocksize.

DEVSTAT_NO_ORDERED_TAGS This device does not support ordered tags.

DEVSTAT_BS_UNAVAILABLE This device supports a blocksize, but it is currently unavailable. This flag is most often used with removable media drives.

Transactions to a device fall into one of three categories, which are represented in the *flags* passed into **devstat_end_transaction**(). The transaction types are as follows:

```
      typedef enum {
                DEVSTAT_NO_DATA       = 0x00,
                DEVSTAT_READ = 0x01,
                DEVSTAT_WRITE           = 0x02,
                DEVSTAT_FREE  = 0x03
        } devstat_trans_flags;
        #define DEVSTAT_N_TRANS_FLAGS   4
```

DEVSTAT_NO_DATA is a type of transactions to the device which are neither reads or writes. For instance, SCSI drivers often send a test unit ready command to SCSI devices. The test unit ready command does not read or write any data. It merely causes the device to return its status.

There are four possible values for the *tag_type* argument to **devstat_end_transaction**():

DEVSTAT_TAG_SIMPLE The transaction had a simple tag.

DEVSTAT_TAG_HEAD The transaction had a head of queue tag.

DEVSTAT_TAG_ORDERED The transaction had an ordered tag.

DEVSTAT_TAG_NONE The device does not support tags.

The tag type values correspond to the lower four bits of the SCSI tag definitions. In CAM, for instance, the *tag_action* from the CCB is ORed with 0xf to determine the tag type to pass in to **devstat_end_transaction**().

There is a macro, DEVSTAT_VERSION that is defined in <*sys/devicestat.h*>. This is the current version of the **devstat** subsystem, and it should be incremented each time a change is made that would require recompilation of userland programs that access **devstat** statistics. Userland programs use this version, via the *kern.devstat.version* **sysctl** variable to determine whether they are in sync with the kernel **devstat** structures.

SEE ALSO

systat(1), devstat(3), iostat(8), rpc.rstatd(8), vmstat(8)

HISTORY

The **devstat** statistics system appeared in FreeBSD 3.0.

AUTHORS

Kenneth Merry <*ken@FreeBSD.org*>

BUGS

There may be a need for **spl**() protection around some of the **devstat** list manipulation code to ensure, for example, that the list of devices is not changed while someone is fetching the *kern.devstat.all* **sysctl** variable.