diskctl(8): A permissively-licensed S.M.A.R.T. and raw disk command utility framework

Michael Dexter
editor@callfortesting.org

The Self-Monitoring, Analysis and Reporting Technology or S.M.A.R.T., is an industry-standard[1] interface implemented by manufacturers of hard disk and solid-state drive devices to present device "health" information to the controlling operating system. This information can include device identification and configuration information, service hours, temperature, failed block reallocation counts, Solid State Disk (SSD) endurance remaining, plus vendor-specific attributes. These attributes are commonly accessed from various operating systems using the "smartmontools"[2] project and specifically the `smartctl(8)` command. While widespread in its use, the smartmontools project does not feature a license that is suitable for inclusion in BSD Unix operating systems, does not support Non-Volatile Memory Express (NVMe) devices, and is limited in its output formatting abilities. The `diskctl(8)` project aims to provide a framework to address the licensing and output formatting limitations of smartmontools and provide a user-friendly framework for new device types and output formatting syntaxes. In addition, `diskctl(8)` aims to provide an interface to common SATA management commands such as `IDENTIFY`, plus the acoustic and power management series of commands. Finally, the `diskctl(8)` project will explore the possibility of supporting VirtIO AHCI S.M.A.R.T. and underlying `zpool(8) status` pass-through for virtual machine disk devices, and other virtualized storage opportunities.

Available S.M.A.R.T. Utility Implementation Advantages and Disadvantages

The prevalent utilities used for accessing S.M.A.R.T. information on the BSD family of operating systems are the `smartctl(8)` utility of the smartmontools project and the `atactl(8)` utilities on OpenBSD and NetBSD.

smartmontools Advantages

The smartmontools project enjoys widespread, if not de facto usage in BSD environments and for good reason:

- Availability on FreeBSD, OpenBSD, NetBSD and DragonFly BSD
- Support for SCSI and SATA devices
- Broad vendor device information including HDDs, SSDs RAID cards and USB bridges
- The included monitoring daemon, `smartd(8)`
- Ability to initiate and terminate basic S.M.A.R.T. tests
- Familiarity brought by widespread use

[1] http://t13.org/
[2] https://www.smartmontools.org/

smartmontools Disadvantages

- Licensing (GPLv2)[3] that precludes its inclusion in BSD operating systems
- Limited output formatting options or per-attribute reporting
- Inability to operate as a library for inclusion in other utilities[4]
- Inability to support NVMe solid-state storage devices
- Inability to perform acoustic management on devices
- Inability to support all security features on devices
- Inability to perform storage bus operations
- Inability to support SATAPI eject
- Inability to support the SCSI Test Unit Ready (`tur`) command
- Inability to deliver raw Command Control Blocks (CCB) to devices
- Inability to support virtual disks[5]

While some of these limitations are addressed by additional in-base and third-party utilities such as FreeBSD's `camcontrol(8)` and the sg3_utils[6] project, each of these demonstrate equally-significant disadvantages.

`atactl(8)` Advantages

- Licensing that allows its inclusion in BSD operating systems
- Inclusion in the OpenBSD and NetBSD operating systems
- Support for acoustic management on devices
- Ability to initiate and terminate basic S.M.A.R.T. tests
- Support for security features on devices
- Support for write cache enable/disable

`atactl(8)` Disadvantages

- Depreciation from the FreeBSD Operating System
- Absent vendor device information
- Limited output formatting options or per-attribute reporting
- Inability to operate as a library for inclusion in other utilities
- Inability to support NVMe solid-state storage devices
- Inability to support SATAPI eject
- Inability to support the SCSI Test Unit Ready (`tur`) command
- Inability to deliver raw Command Control Blocks (CCB) to devices
- Failure to test for S.M.A.R.T. support before performing S.M.A.R.T. commands

[3] https://www.smartmontools.org/browser/trunk/smartmontools/COPYING
[4] https://www.smartmontools.org/ticket/501
[5] https://www.smartmontools.org/wiki/FAQ#Dosmartctlandsmartdrunonavirtualmachineguest0S
[6] http://sg.danny.cz/sg/sg3_utils.html

`diskctl(8)` Licensing Considerations

A firm requirement of the `diskctl(8)` project is that it and its accompanying documentation and vendor information be published until a license that allows its inclusion in the BSD operating systems. The candidate licenses for `diskctl(8)` are the ISC license preferred by the OpenBSD project, the 2-clause BSD license preferred by the FreeBSD project and the MIT license. The MIT license is under consideration to discourage the appearance of favoritism and some thought must be put into the other implications of each candidate license.

Storage Device Interface Considerations

Based on the precedences of the "smartmontools", OpenBSD, FreeBSD and the "sg3_utils" project, the `diskctl(8)` project should target SCSI, SATA and SATAPI devices, plus the more recent NVMe and VirtIO AHCI device types. `diskctl(8)` will be interface-agnostic to the greatest extent possible and establish a clear strategy for incorporating new interface standards. Of these interfaces, the VirtIO interface provides the most interesting area of research, considering that the FreeBSD `virtio_blk(4)` driver can be modified to support passed-through or synthetic S.M.A.R.T. data. This raises interesting questions such as:

- Should Virtual Machines have access to real or synthetic disk health information?
- Should a virtual disk be provided the "worst" S.M.A.R.T. data for all disks on a system?
- Should a virtual disk backed by an aggregate storage device such as a RaidZ ZFS pool be given a synthetic summary of all participating disks and/or the pool's status?
- Should a virtual machine be provided all available disk health data of its host?
- Should a virtual disk provided via a file, iSCSI or NFS provide disk health data?

With regard to virtualization environments such as that provided by the FreeBSD bhyve and OpenBSD vmm hypervisors, it is tempting to assume a "panopticon" approach to security with which virtual machines are provided minimum insights into their host in every regard. This stance is challenged by the fact that system administrator and vendors are increasingly employing hypervisors to give their solutions the ability to perform unobtrusive operating system updates that can be rolled back as needed. Under such a model, any virtual machine is may be considered authoritative over the controlling hardware and thus its storage system. A virtual machine may also obtain the authority to request its own migration to a different host if it determines its backing storage to be a risk. Finally, a consumer of a commodity "cloud" virtual machine provider may have a right to know the health status of their backing store, driving their decision to stay, or move to an alternative host or even service provider.

`diskctl(8)` Library Considerations

The question of virtual disks and the heath of their backing physical disks raises broader questions of what interaction between, or elimination of traditional storage layers should be explored. The OpenZFS file system has been accused of rampant layering violations[7] but those very violations, notably the combination of a volume manger and file system, provide one of its greatest advantages: its ability to rebuild disks using only the user data required,

[7] https://blogs.oracle.com/bonwick/en/entry/rampant_layering_violation

rather than all blocks of a given storage devices. By extension, it is worth considering if the `zpool(8) status` command should incorporate individual disk S.M.A.R.T. status in addition to its current data and metadata block checksum validation status.

The isolated nature of virtual machines and layer-violating nature of the OpenZFS file system raise further questions of their interaction, particularly with regard to plural, averaged and synthetic disk health status reporting. If a given virtualized storage system is provided to a virtual machine or privileged host, it may be justified to provide a list of smart output for all participating disks for parsing by the requesting machine. The traditional S.M.A.R.T. device model however dictates a single response, which could be an average or "worst" response of the member disks. Finally, it is worth exploring if a fully-synthesized S.M.A.R.T. response is justified for virtual disks that takes into account not only underlying disk health but also backing OpenZFS pool status. Ultimately, this is the type of information that must be manually cross-referenced by system operators.

Environment Considerations

We have demonstrated that a case can be made for the aggregation of storage stack health information such as the status information provided by disks and the OpenZFS file system. Field experience will also show that operating system-level data is also critical to accurate disk health monitoring and failure anticipation. Notably:

- A storage device reset count that far exceeds the host operating system or underlying hardware reset count is a reliable indicator of a dysfunctional or failing storage devices controller
- A sudden increase in disk latency is also a reliable indicator of an impending storage device failure

The first of these would require a predictable, cross-operating system means of providing date stamped host reboot information or at a minimum a mechanism for monitoring device resets during a given uptime. This information has demonstrated that when a device's latency begins to increase dramatically, the operator is provided on average between one an eight hours of warning before the final failure of the device. The second of these would require a predictable, cross-operating system means of providing device latency information. This may not be possible without a direct, intrusive latency test but the `dtrace(1)` framework may provide a suitable solution on operating systems that support it.

Not unlike the low-level information that the `dtrace(1)` framework could provide, FreeBSD developer John Baldwin has suggested that a "ccbdump" should be considered to allow operators to observe device Command Control Blocks for debugging purposes in a manner not unlike monitoring network packets. A filtering mechanism could simply report all reads or writes, or provide finer-grained filtering by exact command or content block.

Finally, this research behind `diskctl(8)` has demonstrated that the BSD `dd(1)` utility is inconsistent with regards to its `noerror` option, failing to replace defective blocks with null blocks during a duplication, resulting in inaccurate disk offsets.

`diskctl(8)` Usage Syntax Influences

Unix utilities have long focused on either machine-friendliness as demonstrated by the terse `sysctl(8)` command, or human-friendly `top(1)` utility. Few Unix utilities have been architected to meet the requirements of both human and machine environments, the latter being dominated by administrative shell scripting. At a minimum, the syntax of the `zfs(8)` command will be considered for its ability to list specific properties and for its "scripting mode" and ability to provide parseable values:

```
-H    Used for scripting mode. Do not print headers and separate
      fields by a single tab instead of arbitrary white space.

-p    Display numbers in parseable (exact) values.
```

`diskctl(8)` Output Syntax Framework Considerations

While `smartctl(8)` has focused on human readable output, `camcontrol(8)` '`cmd`' has focused on byte-swapped hexadecimal input and output. System operators are increasingly expecting system output to take the form of JSON, XML, YAML, shell variables and even HTML as system orchestration and "cloud" environments become prevalent. The `diskctl(8)` project will identify a lowest-common denominator "raw" format that is suitable for reformatting into these familiar syntaxes in a manner that is extensible by system operators. This goal should be achievable through unambiguous coding style and both inline and external documentation.

`diskctl(8)` Concept Manual Page

```
diskctl(8)              System Manager's Manual             diskctl(8)

NAME
      diskctl - disk control utility

SYNOPSIS
      diskctl help

      diskctl [-d,a,s,m] <sub command> <device>

      diskctl smart [-o attribute] [-f output format] device

      diskctl led [on|off] device         (Control device LED)

      diskctl cache [-e|-d] device        (Enable|Disable device caching)

      diskctl ccb [-b buffer size] device (Send raw CCB to device)

      diskctl dump [-r|-w] device         (Dump device read|write CCBs)

      diskctl resets device               (Device reset count since boot)

      diskctl eject device                (Eject removable device)

      diskctl (test, secure, acoustic, reset etc.)
```

```
DESCRIPTION

      diskctl allows a system administrator to perform a variety of
      subcommands on a given SCSI, SATA, SATAPI, NVMe or VirtIO block
      storage device.

diskctl [-d,a,s,m] [-o <output format> -Hp] <sub command> <device>

      The diskctl command operates on -s SCSI, -a SATA family and -m NVMe
      devices with default option to -d detect the device type.

      diskctl output options are specified with the -o flag followed by the
      formatting type: human, json, yaml, xml, html, shell.

      -H    Used for scripting mode. Do not print headers and separate
            fields by a single tab instead of arbitrary white space.

      -p    Display numbers in parseable (exact) values.

      The default option is human-readable output.

SUBCOMMANDS

diskctl smart [-o numeric attribute, numeric attribute] device

      Performs a S.M.A.R.T. inquiry on the specified device, reporting all
      supported attributes or a specific list of comma-separated attributes
      prefixed by the -o flag. ...
```

`diskctl(8)` Challenges

Because the respective disk interface commands that `diskctl(8)` must implement adhere to well-established standards such as SCSI and SATA, and the formatting of given responses would involve published standards such as JSON and XML, the greatest challenge of the `diskctl(8)` project will be cross-platform device interface and capabilities identification. At a minimum, `diskctl(8)` should follow the example of the `smartctl(8)` and supplant device auto-identification with the ability to explicitly specify a device type for any given command. i.e. `diskctl -s` for operations on SCSI devices.

Conclusion

Storage devices are the backbone of modern computing and storage systems have long suffered from the limited communication between host bus adapter, storage device, monitoring, partitioning and file system vendors. The `diskctl(8)` project aims to bring the last of these software elements of the storage stack to the BSD and other POSIX operating systems to combat the disunity of these interdependent elements. Over time, the `diskctl(8)` project should provide an effective tool in the ongoing battle against data corruption and loss in computing environments of all sizes.