ATA au Naturel

Erik Quanstrom
quanstro@quanstro.net

ABSTRACT

The Plan 9 *sd*(3) interface allows raw commands to be sent. Traditionally, only SCSI CDBs could be sent in this manner. For devices that respond to ATA/ATAPI commands, a small set of SCSI CDBs have been translated into an ATA equivalent. This approach works very well. However, there are ATA commands such as SMART which do not have direct translations. I describe how ATA/ATAPI commands were supported without disturbing existing functionality.

Introduction

In writing new *sd*(3) drivers for plan 9, it has been necessary to copy laundry list of special commands that were needed with previous drivers. The set of commands supported by each device driver varies, and they are typically executed by writing a magic string into the driver's ctl file. This requires code duplicated for each driver, and covers few commands. Coverage depends on the driver. It is not possible for the control interface to return output, making some commands impossible to implement. While a work around has been to change the contents of the control file, this solution is extremely unwieldy even for simple commands such as IDENTIFY DEVICE.

Considerations

Currently, all *sd* devices respond to a small subset of SCSI commands through the raw interface and the normal read/write interface uses SCSI command blocks. SCSI devices, of course, respond natively while ATA devices emulate these commands with the help *sd*. This means that *scuzz*(8) can get surprisingly far with ATA devices, and ATAPI (*sic.*) devices work quite well. Although a new implementation might not use this approach, replacing the interface did not appear cost effective and would lead to maximum incompatibilities, while this interface is experimental. This means that the raw interface will need a method of signaling an ATA command rather than a SCSI CDB.

An unattractive part of the ATA command set is there are seven protocols and two command sizes. While each command has a specific size (either 28-bit LBA or 48-bit LLBA) and is associated with a particular protocol (PIO, DMA, PACKET, etc.), this information is available only by table lookup. While this information may not always be necessary for simple SATA-based controllers, for the IDE controllers, it is required. PIO commands are required and use a different set of registers than DMA commands. Queued DMA commands and ATAPI commands are submitted differently still. Finally, the data direction is implied by the command. Having these three extra pieces of information in addition to the command seems necessary.

A final bit of extra-command information that may be useful is a timeout. While *alarm* (2) timeouts work with many drivers, it would be an added convenience to be able to specify a timeout along with the command. This seems a good idea in principle, since some ATA commands should return within milli- or microseconds, others may take hours to complete. On the other hand, the existing SCSI interface does not support it and changing its kernel-to-user space format would be quite invasive. Timeouts were left for a later date.

Protocol and Data Format

The existing protocol for SCSI commands suits ATA as well. We simply write the command block to the raw device. Then we either write or read the data. Finally the status block is read. What remains is choosing a data format for ATA commands.

The T10 Committee has defined a SCSI-to-ATA translation scheme called SAT [4]. This provides a standard set of translations between common SCSI commands and ATA commands. It specifies the ATA protocol and some other sideband information. It is particularly useful for common commands such as READ (12) or READ CAPACITY (12). Unfortunately, our purpose is to address the uncommon commands. For those, special commands ATA PASSTHROUGH (12) and (16) exist. Unfortunately several commands we are interested in, such as those that set transfer modes are not allowed by the standard. This is not a major obstacle. We could simply ignore the standard. But this goes against the general reasons for using an established standard: interoperability. Finally, it should be mentioned that SAT format adds yet another intermediate format of variable size which would require translation to a usable format for all the existing Plan 9 drivers. If we're not hewing to a standard, we should build or choose for convenience.

ATA-8 and ACS-2 also specify an abstract register layout. The size of the command block varies based on the size (either 28- or 48-bits) of the command and only context differentiates a command from a response. The SATA specification defines host-to-drive communications. The formats of transactions are called Frame Information Structures (FISes). Typically drivers fill out the command FISes directly and have direct access to the Device-to-Host Register (D2H) FISes that return the resulting ATA register settings. The command FISes are also called Host-to-Device (H2D) Register FISes. Using this structure has several advantages. It is directly usable by many of the existing SATA drivers. All SATA commands are the same size and are tagged as commands. Normal responses are also all of the same size and are tagged as responses. Unfortunately, the ATA protocol is not specified. Nevertheless, SATA FISes seem to handle most of our needs and are quite convenient; they can be used directly by two of the three current SATA drivers.

Implementation

Raw ATA commands are formatted as a ATA escape byte, an encoded ATA protocol proto and the FIS. Typically this would be a H2D FIS, but this is not a requirement. The escape byte 0xff, which is not and, according to the current specification, will never be a valid SCSI command, was chosen. The protocol encoding proto and other FIS construction details are specified in /sys/include/fis.h. The proto encodes the ATA protocol, the command size and data direction. The atazz command format is pictured in Figure 1.

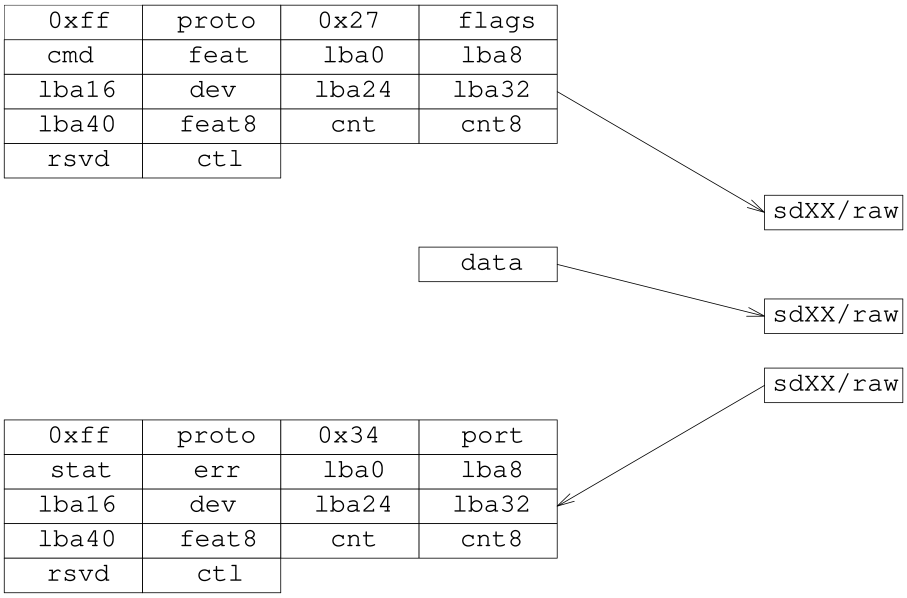

Figure 1

Raw ATA replies are formatted as a one-byte sd status code followed by the reply FIS. The usual read/write register substitutions are applied; ioport replaces flags, status replaces cmd, error replaces feature.

Important commands such as SMART RETURN STATUS return no data. In this case, the protocol is run as usual. The client performs a 0-byte read to fulfill data transfer step. The status is in the D2H FIS returned as the status. The vendor ATA command 0xf0 is used to return the device signature FIS as there is no universal in-band way to do this without side effects. When talking only to ATA drives, it is possible to first issue a IDENTIFY PACKET DEVICE and then a IDENTIFY DEVICE command, inferring the device type from the successful command. However, it would not be possible to enumerate the devices behind port multiplier using this technique.

Kernel changes and Libfis

Very few changes were made to devsd to accommodate ATA commands. the SDreq structure adds proto and ataproto fields. To avoid disturbing existing SCSI functionality and to allow drivers which support SCSI and ATA commands in parallel, an additional ataio callback was added to SDifc with the same signature as the existing rio callback. About twenty lines of code were added to port/devsd.c to recognize raw ATA commands and call the driver's ataio function.

To assist in generating the FISes to communicate with devices, libfis was written. It contains functions to identify and enumerate the important features of a drive, to format H2D FISes And finally, functions for *sd*s and *sd*-devices to build D2H FISes to capture the device signature.

All ATA device drivers for the 386 architecture have been modified to accept raw ATA commands. Due to consolidation of FIS handling, the AHCI driver lost 175 lines of code, additional non-atazz-related functionality notwithstanding. The IDE driver remained exactly the same size. Quite a bit more code could be removed if the driver were reorganized. The mv50xx driver gained 153 lines of code. Development versions of the Marvell Orion driver lost over 500 lines while libfis is only about the same line count.

Since FIS formats were used to convey commands from user space, libfis has been equally useful for user space applications. This is because the *atazz* interface can be thought of as an idealized HBA. Conversely, the hardware driver does not need to know anything about the command it is issuing beyond the ATA protocol.

Atazz

As an example and debugging tool, the *atazz* (8) command was written. *Atazz* is an analog to *scuzz* (8); they can be thought of as a driver for a virtual interface provided by *sd* combined with a disk console. ATA commands are spelled out verbosely as in ACS-2. Arbitrary ATA commands may be submitted, but the controller or driver may not support all of them. Here is a sample transcript:

```
az> probe
/dev/sda0     976773168;   512   50000f001b206489
/dev/sdC1     0;  0     0
/dev/sdD0     1023120;   512     0
/dev/sdE0     976773168;   512   50014ee2014f5b5a
/dev/sdF7     976773168;   512   5000cca214c3a6d3
az> open /dev/sdF0
az> smart enable operations
az> smart return status
normal
az> rfis
00
34405000004fc2a00000000000000000
```

In the example, the probe command is a special command that uses #S/sdctl to enumerate the controllers in the system. For each controller, the sd vendor command 0xf0 (GET SIGNATURE) is issued. If this command is successful, the number of sectors, sector size and WWN are gathered and and listed. The /dev/sdC1 device reports 0 sectors and 0 sector size because it is a DVD-RW with no media. The open command is another special command that issues the same commands a SATA driver would issue to gather the information about the drive. The final two commands enable SMART and return the SMART status. The smart status is returned in a D2H FIS. This result is parsed the result is printed as either normal, or threshold exceeded (the drive predicts imminent failure).

As a further real-world example, a drive from my file server failed after a power outage. The simple diagnostic SMART RETURN STATUS returned an uninformative threshold exceeded. We can run some more in-depth tests. In this case we will need to make up for the fact that *atazz* does not know every option to every command. We will set the lba0 register by hand:

```
az> smart lba0 1 execute off-line immediate # short data collection
az> smart read data
col status: 00 never started
exe status: 89 failed: shipping damage, 90% left
time left: 10507s
shrt poll: 176m
ext poll: 19m
az>
```

Here we see that the drive claims that it was damaged in shipping and the damage occurred in the first 10% of the drive. Since we know the drive had been working before the power outage, and the original symptom was excessive UREs (Unrecoverable Read Errors) followed by write failures, and finally a threshold exceeded condition, it is reasonable to assume that the head may have crashed.

Stand Alone Applications

There are several obvious stand-alone applications for this functionality: drive firmware upgrade utility, drive scrubber that bypasses the drive cache and a SMART monitor.

Since SCSI also supports a basic SMART-like interface through the SEND DIAGNOSTIC and RECEIVE DIAGNOSTIC RESULTS commands, *disk/smart* (8) gives a chance to test both raw ATA and SCSI commands in the same application.

Disk/smart uses the usual techniques for gathering a list of devices or uses the devices given. Then it issues a raw ATA request for the device signature. If that fails, it is assumed that the drive is SCSI, and a raw SCSI request is issued. In both cases, *disk/smart* is able to reliably determine if SMART is supported and can be enabled.

If successful, each device is probed every 5 minutes and failures are logged. A one shot mode is also available:

```
chula# disk/smart -atv
sda0: normal
sda1: normal
sda2: normal
sda3: threshold exceeded
sdE1: normal
sdF7: normal
```

Drives sda0, sda1 are SCSI and the remainder are ATA. Note that other drives on the same controller are ATA. Recalling that sdC0 was previously listed, we can check to see why no results were reported by sdC0:

```
chula# for(i in a3 C0)
    echo identify device |
        atazz /dev/sd$i >[2]/dev/null |
        grep '^flags'
flags   lba llba smart power nop sct
flags   lba
```

So we see that sdC0 simply does not support the SMART feature set.

Further Work

While the raw ATA interface has been used extensively from user space and has allowed the removal of quirky functionality, device setup has not yet been addressed. For example, both the Orion and AHCI drivers have an initialization routine similar to the following

```
newdrive(Drive *d)
{
        setfissig(d, getsig(d));
        if(identify(d) != 0)
                return SDeio;
        setpowermode(d);
        if(settxmode(d, d->udma) != 0)
                return SDeio;
        return SDok;
}
```

However in preparing this document, it was discovered that one sets the power mode before setting the transfer mode and the other does the opposite. It is not clear that this particular difference is a problem, but over time, such differences will be the source of bugs. Neither the IDE nor the Marvell 50xx drivers sets the power mode at all. Worse, none is capable of properly addressing drives with features such as PUIS (Power Up In Standby) enabled. To addresses this problem all four of the ATA drivers would need to be changed.

Rather than maintaining a number of mutually out-of-date drivers, it would be advantageous to build an ATA analog of pc/sdscsi.c using the raw ATA interface to submit ATA commands. There are some difficulties that make such a change a bit more than trivial. Since current model for hot-pluggable devices is not compatible with the top-down approach currently taken by *sd* this would need to be addressed. It does not seem that this would be difficult. Interface resets after failed commands should also be addressed.

Source

The current source including all the pc drivers and applications are available in the following *contrib* (1) packages on *sources*:
quanstro/fis,
quanstro/sd,
quanstro/atazz, and
quanstro/smart.

The following manual pages are included:
fis (2), *sd* (3), *sdahci* (3), *sdaoe* (3), *sdloop* (3), *sdorion* (3), *atazz* (8), and *smart* (8).

Abbreviated References

[1] *sd* (1), published online at http://plan9.bell-labs.com/magic/man2html/3/sd.

[2] *scuzz* (8), published online at http://plan9.bell-labs.com/magic/man2html/8/scuzz.

[3] T13 *ATA/ATAPI Command Set 2*, revision 1, January 21, 2009, formerly published online at http://www.t13.org.

[4] T10 *SCSI/ATA Translation 2 (SAT 2)*, revision 7, February 18, 2007, formerly

published online at http://www.t10.org .