Devsd Refresh

Erik Quanstrom
quanstro@quanstro.net

ABSTRACT

The Plan 9 *sd*(3) interface provides uniform access for disk-like devices for Plan 9. Its interfaces have proven fairly robust. With no incompatible changes, features like full 48-bit (ATA) and 64-bit (SCSI), SGPIO[1]/SES-2[2] enclosure management lights, and centralized SCSI CDB to ATA translation have been straightforward to add. Nonetheless, *sd* assumes fixed rather than hot-swappable drives and uses a intricate drive lettering scheme that may require some client-visible changes in the future.

Introduction

Although *sd*(3) stands for storage device, it belies its SCSI roots. Nonetheless, it has proved quite robust in the face of changing requirements and easily extensible. The addition of several new hot-pluggable SATA and combined mode SATA/SAS controllers has stretched the model a bit. New drives and controllers already can break the 32-bit LBA barrier. Certainly this will be common in the future. Revisions to the ATA standard have introduced tricky new power-saving modes. We will outline the changes that were necessary to support these new drivers, the changes to the drivers themselves and what needs to be addressed in the future.

Raw ATA Support

While Plan 9 allows user-level commands to send raw SCSI commands directly to devices through the `raw` file, there is no such facility for ATA. Drives using the ATA command set get by by emulating a small set of SCSI commands. As outlined in the *ATA au Naturel*[3] paper, support for raw ATA commands was added without disturbing existing SCSI functionality. This required an extra *ataio* function and added about 20 lines of code to the generic device.

Full LBA Support

Full LBA support has been limited at several points to 32 bits. This limits drive size to 2TB with 512-byte sectors. With drive sizes currently at 2TB and chipset-level RAID that can combine drives, this is becoming a serious limitation. While *sdaoe*(3) sidesteps many of these issues, the other drivers have not been addressed.

These 32-bit limitations are in part due to the translation to SCSI in `pc/sdscsi.c`. The current ATA drivers all translate IO requests into SCSI CDBs and then translate them from SCSI to the original LBA and number of sectors. So SCSI limitations also limit ATA. Finally, ATA drivers are unable to translate `READ/WRITE (16)` back into LBA format.

The initial step, generating the 16-byte IO commands when necessary was taken several years ago. However, drivers using `scsionline` were still limited to 32-bits due to the use of `GET CAPACITY (10)`. It was straightforward to detect overflow and issue a 16-byte command. As we consider the drivers, the translation from CDB to LBA in the various ATA drivers (*sdaoe* excepted) do not translate `READ/WRITE (16)`. Rather than fix them all individually, a new function *sdfakescsirw* was added which converts a CDB back into an LBA, number of sectors and if it is a read or write request. This fixed the 32-bit limitation and removed a number of duplicate (and not entirely compatible) copies of the same code.

For many drivers, it does not make sense to translate from LBA to CDB and back again. The *sdiahci* driver was modified to take advantage of this. A new *ahcibio* function was written that uses the LBA, sector count directly. In the case of ATAPI devices only, a SCSI CDB is generated. For *sdaoe* the new *aoebio* is even simple, since the underlying protocol does not support ATAPI.

Unfortunately, a *rio* function must still be provided to support basic *scuzz(8)* functionality. This limits the amount of code that can be eliminated.

Lights

SES-2 is a standard defining how to interact with a drive enclosure. Features vary widely by enclosure and HBA. It is possible to control up to three lights per drive (a green activity light, an amber locate light and a red fail light). In addition, a number of sensor readings may be available.

Rather than add add-hoc functionality to *sd*, it was decided to pass SGPIO/SES-2 commands through via control files. It would have been possible to simply reuse the `sd`*XX*`/ctl` file, but this would have made detecting and parsing the status of the lights cumbersome. Instead, a facility to add extra control files to a device was added. A single function was added to register a new control file.

For SGPIO/SES-2 lights, a `led` control file was added. Reading this file returns the current slot state. This is one of normal, rebuild, locate, spare, or fail. Writing a new state to the `led` file sets that slot to the given state. For example

```
chula# cat led
normal
chula# echo fail>led
```

sets the fail light on solid red and turns on the backplane alarm. To reset the alarm,

```
chula# echo normal>led
chula# cat led
normal
```

It is envisioned that this functionality could be harnessed by *fs*(3) or *smart*(8) to automaticly announce the state of drives.

Autosense

There are two ways for SCSI commands to return data. The method used depends on the HBA and the SCSI transport protocol. Early SCSI standards required the `REQUEST SENSE` to return sense (error) data. Autosense is when the sense data is returned along with the response. When autosense is used, the sense data remains clear. So a second `REQUEST SENSE` would return SAM-3[4] makes autosense mandatory. However the protocol provided by the `raw` file requires an explicit `REQUEST SENSE`. To get around this, *sd* uses the flag `SDnosense` to force the driver to hold on to sense data.

Since this forces an additional burden on most modern drivers, the technique used by *sdmylex* of saving sense data was moved to *sd*. Conversion to the new system remains unfinished as no working machines capable of using 5V PCI cards were available for testing. The old sense-data saving code will simply be unused.

Bits

A few odds and ends were also changed. According to the *sd* manual page, the 10th unit attached to controller 0 should be named `sd0a`. Unfortunately, it was named `sd010`. In this case, the manual was considered authoritative and the code was "fixed."

It had also been impossible for anyone other than *eve* to read the `sdctl` file. Since the list of of controllers is not a security concern, the ability for *eve* to grant looser permissions was added in line with other *sd*-generated files. In addition, the error messages were sharpened so that setting unsettable information with *wstat* is flagged as an error.

There was also a vendor-specific `MODE SENSE` response which could be used to return the `IDENTIFY (PACKET) DEVICE`data. It appeared that this wormhole was unused, except by *scuzz*(8). It was removed since this data is accessible more conventionally via *atazz*(8).

The `sdctl` file is now generated so that there is exactly one line per drive letter. Formerly, a line was generated only if the driver provided an appropriate *rtopctl* function. *Sd* now supplies a basic entry for drivers lacking this function. This simplifies unit enumeration by spelling out each controller prefix. For each prefix, only 16 units need to be probed.

Drivers

The following drivers have received significant work, or are new: *sdaoe*, *sdata*, *sdiahci*, *sdloop*, *sdorion*, *sdmv50xx*. The *sdloop* (loopback) and *sdorion* drivers are new. All were converted to use `libfis`. Print statements were audited so drivers properly identify themselves in console messages.

The *Sdata* driver now uses the standard *sdsense*, *sdfakescsi* and *sdfakescsirw* functions. Additionally, a bug that caused Intel ICH south bridges to hang the system on boot was fixed.

The AHCI driver was updated to revision 1.3 of the specification[5] and power management support was added. Support for sector sizes other than 512 bytes was added. Disabled ports, staggered spinup, device discovery ATAPI devices have all been fixed. PATA bridged drives such as SATA Disk on Modules (DOMs) are now supported. Several AMD SB6xx bugs were fixed and support for VIA, nVidia, and JMicron devices was added.

Further Work

While all of the changes to *sd* could use further polish, there are two areas that have been ignored: hot-pluggable devices and drive enumeration. Hot-pluggable devices are a rough fit in *sd* because they may appear at any time and disappear at any time. The parallel SCSI drivers only create units where a drive is detected at boot time. The IDE driver takes the same approach. Since it is assumed that drives are not hot-pluggable, this makes sense. (As long as the IDE driver is actually presenting PATA drives, that is.) For SATA and SAS, this approach doesn't work. Either device directories must be created dynamicly, or a directory for each port must be created on boot. The later approach was taken, though this may need revisiting in the face of SAS expanders and

SATA port multipliers.

Another consequence of hot-pluggable drives is that it's hard to know how long to wait at boot for drives to appear. In a large system with many drives, it's not uncommon for drives to straggle in several minutes after the operating system gains control. Clearly it would be unacceptable to wait two minutes for each drive. But if that drive contains your root filesystem, you need to wait. The current kludge is to use *9load* use a heuristic algorithm to spin up the drives it sees. This seems to be working well in practice, but does not appear to be a satisfactory solution.

The *verify* and *online* model for bringing a unit online seem at odds with hot-plug devices. An experimental version of *sd* was built that combined the initial *pnp*, *enable* and *verify* functions into *pnp*. This allows drive discovery to take place in parallel and reduces the amount of state the controller needs to carry. The *online* function was replaced with *ready*. This function returns 1 if the drive is ready for access. The process of bringing the drive online is done asynchronously by a background process. As this is already a requirement for hot-pluggable drives, this requires no addition code. While this approach has worked well and simplified the code, it still doesn't answer the question of when to wait for the drive to be ready and for how long.

Currently *sd* picks meaningful drive letters. On a pc, `sdC0` and `sdD0` are, as expected, the primary IDE master and the secondary IDE master. However, if these same drives are not given the IDE legacy IO ports, they would be labeled `sdE0` and `sdF0`. Other ATA drives start with drive letter 'E.' SCSI drives start with the first controller being named `sd0`. The new Orion driver can control either SATA or SAS drives. The only solution in the current scheme is to start with a new primary letter — 'a' was chosen. My primary development machine has drive letters a, C—F, and l. This is a somewhat unwieldy situation. It may make sense to enumerate units sequentially. So instead of having units `sda0` and `sdE0` one would have `sd00` and `sd01`. If it is worth preserving the ability to associate drive letters with controllers, sdctl can continue to provide the mapping for interested applications. Dynamicly configured drives could read their assigned letters back.

Abbreviated References

[1] Serial GPIO, published online at
`ftp://ftp.seagate.com/sff/SFF-8485.PDF`.

[2] SCSI Enclosure Services – 2 (SES-2), published online at
`http://www.t10.org/ftp/t10/drafts/ses2/ses2r19a.pdf`

[3] E. Quanstrom "ATA au Naturel", proceedings of the International Workshop on Plan 9, October, 2009.

[4] SCSI Architectural Model – 3 (SAM-3), published online at
`http://www.t10.org/cgi-bin/ac.pl?t=f&f=sam3r14.pdf`

[5] Advanced Host Controller Interface (AHCI) 1.3, published online at
`http://download.intel.com/technology/serialata/pdf/rev1_3.pdf`