Sector Size Issues with OS-9 and RS-DOS

Boisy G. Pitre

Introduction

In the late 1980's, the advent of SCSI controllers for the Color Computer have created a demand for all types of SCSI devices to be interfaced. Early on, SCSI hard drives supported 256 byte sectors, which worked well with OS-9 and RGB-DOS, but as time went on, sector sizes increased to 512 bytes and beyond. This caused compatibility problems with both OS-9 and RGB-DOS, problems which have been solved in a number of different ways.

The 256 Byte Sector Limitation

The specific problem that is caused by sector sizes greater than 256 bytes is manifested in the inherent design of both OS-9's RBF file manager and RS-DOS itself. In both cases, the software was written to pass 256 byte sector buffers to and from applications. In turn, these applications were usually written with hard knowledge that a sector is always 256 bytes.

In the case of RBF, its very data structures and interface into the driver depend upon the fact that the sector buffer is 256 bytes. To adapt RBF to different sector sizes would necessitate a fundamental change the file manager and the file system, which would break existing disk based applications.

Approach 1: Deblocking

In the OS-9 world, there have been several software work-arounds developed to overcome RBF's insistence that a sector is 256 bytes. The most popular and effective is the "deblocking method."

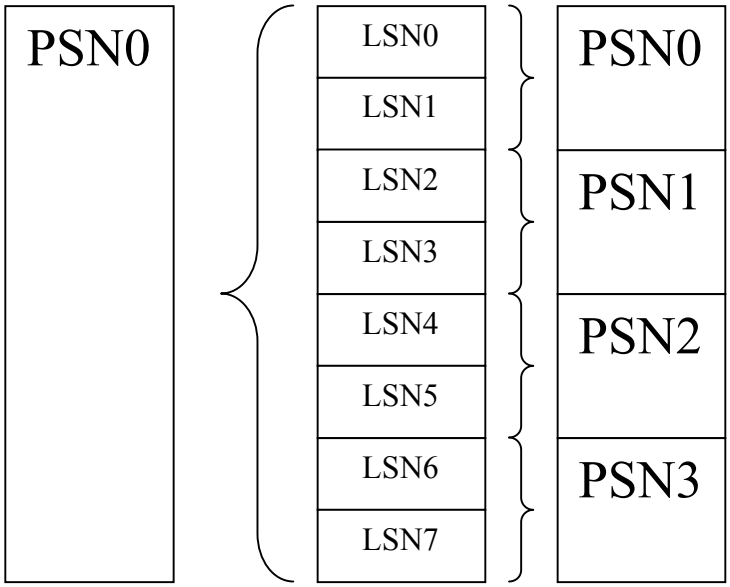
The deblocking method works on the premise that since the RBF file manager needs 256 byte sectors, then it is up to the driver to properly present all sector data in this way. This puts the burden on the driver itself, and leaves RBF and applications to blindly see things in the 256 byte sector prism.

The approach is simple: take the Logical Sector Number (LSN) that RBF gives the Read/Write routine of the driver, divide it in some part, and use the quotient, known as the Physical Sector Number (PSN), to find the appropriate 256 byte sector.

The following illustration shows the relationship between the logical sector number and physical sector number. If RBF asks for LSN2 on a 512 byte sector device, then the driver must request PSN1 from the device and return the upper 256 bytes to RBF. Additionally, if RBF requests LSN3, the driver must request PSN1 again from the device and return the lower 256 bytes to RBF.

However in the case of a 2,048 byte sector device (CD-ROMs usually have this sector size), any access to LSN0-7 requires the driver to read PSN0 and select the appropriate 256 byte block from there.

In the case of a device with 512 byte sectors, there is a 2:1 mapping between LSN and PSN. For 2,048 byte devices, there is an 8:1 mapping.



Read Deblocking

Read deblocking is fairly simple. Using the following formula, we can obtain the proper PSN for a given LSN. The sector size (SS) must also be provided:

PSN = LSN / (SS / 256)

Example 1: RBF requests LSN 5 from a device with a Sector Size (SS) of 512 bytes.

PSN = 5 / (512 / 256)
PSN = 5 / 2
PSN = 2

The PSN is 2. Now we compute which 256 byte block in the 512 byte PSN is to be returned to RBF by obtaining the modulo of the LSN and SS

BLOCK = LSN % (SS / 256)
BLOCK = 5 % (512 / 256)
BLOCK = 5 % 2
BLOCK = 1

So, LSN 5 is found in PSN 2, block 1.

Example 2: RBF requests LSN 8 from a device with a Sector Size (SS) of 512 bytes.

PSN = 8 / (512 / 256)
PSN = 8 / 2
PSN = 4

The PSN is 4. Now we compute which 256 byte block in the 512 byte PSN is to be returned to RBF by obtaining the modulo of the LSN and SS

BLOCK = LSN % (SS / 256)
BLOCK = 8 / (512 / 256)
BLOCK = 8 % 2
BLOCK = 0

So, LSN 8 is found in PSN 4, block 0.

Example 3: RBF requests LSN 26 from a device with a Sector Size (SS) of 2,048 bytes.

PSN = 26 / (2048 / 256)
PSN = 26 / 8
PSN = 3

The PSN is 3. Now we compute which 256 byte block in the 2,048 byte PSN is to be returned to RBF by obtaining the modulo of the LSN and SS

BLOCK = LSN % (SS / 256)
BLOCK = 26 / (2048 / 256)
BLOCK = 26 % 8
BLOCK = 2

For this example, LSN 26 is found in PSN 3, block 2.

Write Deblocking

Writing to a non-256 byte sector device is slightly more involved than reading from one. The reason is that the sector must be written at once. In the case of a 512 byte sector

device, all 512 bytes must be presented to the device in order for that sector to become updated.

The dilemma here is that in the Write routine, just as in the Read routine, all we are presented with is a 256 byte buffer.

In order to honor RBF's request and update the 256 byte block, we must read in an ENTIRE 512 byte sector, update the appropriate block in that sector, and write out the ENTIRE 512 byte sector to the device.

Using the previous illustration as a guide, let's suppose that RBF asks us to write LSN 3 onto a 512 byte sector device. In order to do this, we must use the formulas already discussed to find the PSN. We must then read the ENTIRE PSN into memory, and using the block formula, determine which 256 byte block in our PSN buffer should be updated with our 256 byte buffer we've been given by RBF. We must then copy the contents of that 256 byte buffer into the appropriate offset of the PSN buffer, and write the entire 512 bytes out to the PSN location we computed earlier.

Performance Penalties

It should be obvious by now that in both the read and write case, there are performance penalties associated with deblocking. In the read case, we may have to read up to N-1 256 byte blocks before obtaining the desired block for RBF. In the case of writing, we have no choice but to read an entire PSN just to update a single 256 byte block, and then write the entire PSN back to the device.

Approach 2: First Block Only

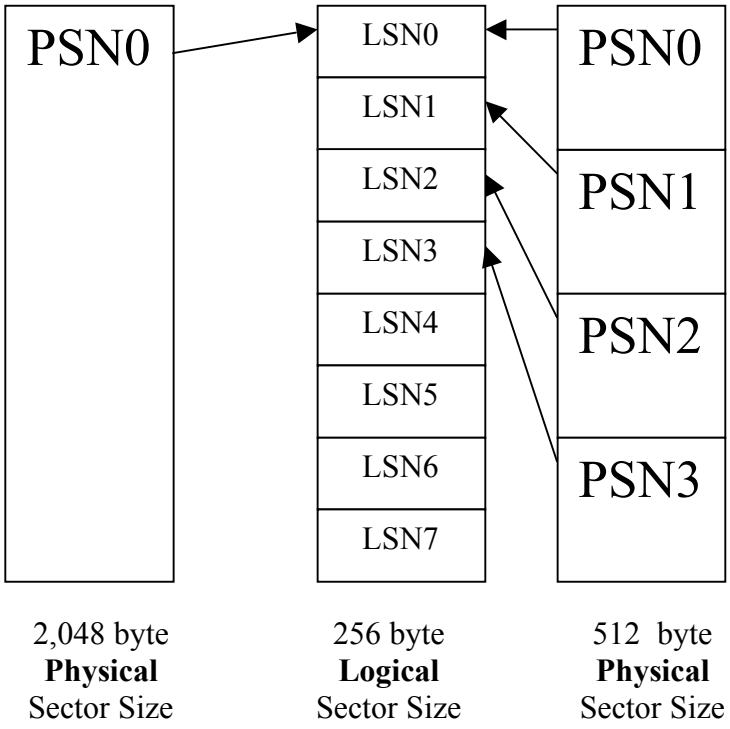
Another approach to use in dealing with sector sizes greater than 256 bytes is the *First Block Only* approach. That is, only the first 256 bytes of a PSN is used by the system, and any remaining blocks in that PSN are wasted.

The appeal to this approach is simplicity and speed. There is no need to do computations to calculate block offsets for reading or writing. The down-side is that in the best case of 512 bytes per sector, half of a device's space is wasted.

The following illustration shows the relationship between 512 and 2,048 byte physical sector sizes to a 256 byte logical sector size when the First Block Only approach is used.

In the case of the 2,048 block PSN0, the entire 2,048 bytes are used to house LSN0, which is just 256 bytes, yielding an unused count of 2,048-256, or 1,792 bytes. This means 7/8 of this sector will be wasted and hold no meaningful information.

Likewise, the 512 byte PSN0 on the right points to LSN0, the 512 byte PSN1 points to LSN1, and so on. This shows clearly a 1:1 mapping between the physical and logical sectors, but with a waste factor of 50%.


PSN0
LSN0
LSN1
LSN2
LSN3
LSN4
LSN5
LSN6
LSN7
PSN0
PSN1
PSN2
PSN3
2,048 byte
Physical
Sector Size
256 byte
Logical
Sector Size
512 byte
Physical
Sector Size