OVMS Investigation Report for SCSI Naming

Digital Confidential

Glenn C. Everhart

15–April–1997

Abstract

This project plan reports results of investigation into how OpenVMS SCSI naming needs to be changed to handle imminent changes to what it serves.

1 The Problem

The current SCSI subsystem uses the device hardware address as a name, using a single letter for the port, and using ID*100 + LUN as the unit number.

Both of these parts of SCSI device names are becoming inadequate in range in that there are already systems (Turbolasers for example) which can be equipped electrically with more than 26 SCSI busses, and the new fibre channel SCSI busses can address potentially thousands of devices.

Moreover, the ID and LUN used to compute a unit number are becoming obsolete, and are being replaced with wide (64 or 128 bits at the moment) world wide IDs. This will make the present "geographical" naming scheme unworkable.

For these reasons, work which will increase the number of ports which can be addressed and the number of devices to be addressed is essential.

2 Goals:

- Increase the number of SCSI ports usable to well beyond 26.
- Increase total devices usable greatly.
- Have names which are stable from boot to boot and usable to customers.
- Avoid laying configuration burdens on customers.
- Permit some selectivity in what is configured.
- Support finding which device has what name.

3 Non-Goals:

- Devise details of how one sets up names when autoconfiguring a fibre channel.
- Specify all details of how SCSI devices are found on a bus.

4 Background:

OpenVMS device names cannot exceed 15 characters; that limitation is present in too much code to change now. This completely prevents a simple encoding of a world wide ID as a device name. Moreover, various pieces of code assume a two letter device name, and for cluster - known devices that there is a controller letter and unit number.

5 Approaches:

The approaches will be discussed in two parts: one for the number of ports issues, and another for the naming of devices attached to those ports.

5.1 Port Usage:

5.1.1 Notion I: Multiletter port "letters" coupled to class names

This is not the chosen approach.

At present, a coupling between port driver name and class driver name is maintained by using the same controller letter. Were the boot path altered to use two letters for ports, and this carried over into class devices, up to 702 ports could be addressed. This is probably enough for the foreseeable future, and the modifications needed to INIT_IO_DB and friends appear not too difficult. Consoles already use multiletter port letters, avoiding possible confusion there.

Were class devices to be named with more letters for port number, though, there would be less characters for unit number, a spot also in need of growth.

5.1.2 Notion II: Multiletter port "letters" uncoupled.

This is the chosen approach.

A correspondence between port letter and class device names is however not essential. Port drivers are visible only in the processors they are connected to, not cluster wide, so there is no need for their names to be coordinated. So long as a cross index is kept somewhere to show which port driver is connected to what devices, the information can be obtained at need without constraining names.

Class device units however must have consistent naming cluster wide, but need not have any fixed port letter, provided there is other means to find the port driver. (This means already is implicit in the OpenVMS I/O data base, although displays for it would need to be made available.)

5.1.3 Other Approaches Not Chosen:

Using unit numbers for port drivers or using port allocation classes for port drivers have been considered and rejected. The first notion however would make it necessary to group port drivers by type, which the boot code has no notion of at all now. The second would require port allocation classes to be assigned very early in boot. They could not be made cluster consistent, which makes them questionably useful, and would violate their meaning elsewhere.

5.1.4 Discussion of the Chosen Approach:

The approach favored is to allow multiple letters to be used for port driver port numbers, since these devices are not known outside their own CPU anyway, and define the names of class drivers separately. This will mean only minor alteration to the boot path and allows naming of other devices to be deferred (except for boot devices) until the time class driver units are connected. At this time, in a cluster, a distributed lock manager exists and can assist in providing cluster consistency.

As a side remark, it could be handy to have the port unit names stable across boots for debugging purposes. Unfortunately, OpenVMS does not keep any "hardware path" information around which could be used to make port names more stable than the order they are found in scan. However, by divorcing port driver names from class driver names we gain the advantage as well that device names will depend only on their worldwide ID, not on their port.

5.2 A Bit of Supporting Detail for Port Approach:

In init_io_db, a full longword is available for the port "letter", so that the progression A,B,C...Z,AA,AB,AC,...AZ,BA,BB,...ZZ seems feasible. There are assumptions that the name length is 4 characters scatterd here and there, but if the unit number parsing is arranged so that a name like PKAC: is treated as though it has been PKAC0: (with all unit numbers being zero) this can be lived with for the currently foreseeable future. This encoding can handle up to 702 ports per choice of the initial two letters, enough for machines we have and likely enough for some time to come. (By using 3 letters, the number of possibilities grows to 18954, should the 4 letter name assumptions in init_io_db be relaxed. Enough other assumptions in VMS file handling would need to be updated to handle these numbers of controllers that it seems unlikely that it will be necessary to exceed that for at least the next 5 years.)

5.3 Unit Usage:

A number of possible approaches exist to handling unit names also. It is a reasonable limitation to restrict consideration here to cases where worldwide IDs are available, since only in those cases does the current naming scheme break down.

Several possible approaches were outlined in John Hallyburton's IR on SCSI naming. I will not repeat those arguments here.

There are two possibilities which appear to have some merits and one other which deserves mention.

5.3.1 Notion I: Hash Coded WW IDs

This is not the chosen approach.

Given a worldwide ID, one could conceivably use a hash code to convert the ID into something that would fit into 5 digits of unit number, or perhaps 5 digits of unit number, a port letter, and perhaps 4 digits of allocation class.

There are two problems:

1. The hash codes cannot be guaranteed unique, so some kind of cluster arbitration is necessary to check for collisions. Thus no speed advantage is to be had over other systems for naming, and the resolution of collisions would have to be made stable somehow.
2. Device names in this system would be well and truly scattered all over the name space, with neither rhyme nor reason and forcing customers to type long names, or define logicals to point at them, an unacceptable system administration burden.

These make this approach unworkable.

5.3.2 Notion II: Port Allocation Classes Everywhere

This is not the chosen approach.

If every bus on a system has a Port Allocation Class, device unit assignment on the bus will be cluster unique with no need to cooperate to assign names beyond the direct connected system(s). A file of common port allocation class names would have to be maintained the same globally (possibly as it is now) and in addition configuration files would need to be kept in synchronization between any processors sharing a bus (and for processors with

different paths to the same device). This synchronization (and that for global port allocation classes) can be done with the lock manager using the scheme to be defined below. It has the advantage that once the PACs are set up, coordination traffic for device naming needs to be done only between nodes where busses are shared.

The system would initially create a configuration file for each bus which would have tuples containing worldwide ID for a device and naming information (e.g., a port letter and a unit number, so that the total namespace would hold 26 * 65536 units. Unit numbers must fit in 16 bits to fit into existing UCB definitions.). This configuration file would be initially read in at boot time to provide name stability across boots and would be arbitrated at each boot to ensure cluster commonality.

This system would work (there are at least 32767 allocation classes available, enough to cover any at all probable collection of busses in a cluster). It would be largely automatic in setup and operation, and the configuration file could be edited to allow selection of some devices and not others to actually be configured. Its cluster traffic reduction is an advantage, but deemed a minor one for any currently foreseeable configuration sizes. More efficient cluster communications could, in addition, render it moot even in larger ones.

The disadvantage the system has is that it perpetuates the use of allocation classes which will appear random to the customer and will be more difficult to use than might be attained. Also it means the unit assignment will be a function of pure history, and again not be easily altered, and the device name's form will be cast more deeply in concrete to what it is today than before. It seems preferable to allow device names to be simpler.

5.3.3 Notion III: Configuration Files Arbitrated By Lock Manager

This approach seems best.

When we set up class devices, we will have the cluster lock manager present (except for the boot device, about which more later). Therefore it is feasible to use the lock manager to arbitrate any names we care to use. (Higher bandwidth intracluster communications might be used if needed in the future as well. These would also be available by the time they are needed in the boot.) The "unit number" namespace can be considered to be the port letter plus the unit number characters as a "phase I" implementation, and device names can be assigned sequentially, with WW ID to device name being recorded in a configuration file the first time used, and read in thereafter. The configuration files would be consistency checked clusterwide as a cluster boots, and made consistent if they were not initially so, so that the recorded configuration files would quickly be converged to being identical even if they were not the same at first.

The configuration file and arbitration scheme can be extended to allow some devices not to be configured, and the device name chosen need not always be what was initially selected. A configuration file could permit the chosen name of a device to be anything the customer wanted and a checking system would ensure it was unique. For Phase I, however, we will consider only names which follow the traditional VMS naming scheme: 2 letter device name, 1 port letter, and up to 5 digit unit number. These names will be arbitrated to be cluster unique and the same for the same WW ID, which will largely make port allocation classes unnecessary. By arbitrating in this way and making use of the fact that the device WW ID supplies a sure way to tell when multiple paths exist to a device, we can also avoid the need for SCSI target mode in SCSI clusters provided a connection guarantees WW IDs.

It might be added that console supplied device "nicknames" can be handled in newer VMS consoles. These might be used also, provided some way to handle or prevent conflicts between multiple consoles can be devised.

For boot devices, the checking described below cannot assign a name, but can ensure it is consistent. If it is not, it will print an error message and exit. A SYSGEN parameter should be used to allow the boot device name to be externally specified. This will work similarly to the existing code which is used for shadowing.

This approach seems simplest both to implement and for customers to use.

6 Locking Protocol For Path / Name Resolution

6.1 Supporting Detail for Unit Approach

The locking protocol needed to read a configuration file and ensure that its results match clusterwide is about the same whether matching a pair of pathnames on one node with the pair on another, a port and unit number and WW ID on one node and the same information on another, or a hardware connector number and port "letter" and global ID on one node and the same information on another. Therefore let us sketch a protocol for making this information consistent.

1. A node starting enqueues a "naming master" lock. If it receives this lock, it is the naming master for the cluster and hangs onto the lock basically forever. If not, it gets the lock information for the master so it can find where the master is.
2. For every naming master lock, there is a "right to communicate" lock. A non master delays briefly, and after this possible delay each node attempts to obtain the "right to communicate" lock. The master will initially get it because it will not delay, and as a failsafe it will fill the lock value block with its own ID as a flag to all other nodes that it has had a chance to initialize. The master will read its configuration file first and set up its in-memory database of naming information or of path information, will perform its local device scan and perform any needed connects, and have its database ready for queries from other nodes prior to releasing the "right to communicate" lock, so that acquiring this lock will generally mean that another node can communicate now. (If the naming master crashes and another processor gains the lock, other processors must check that the lock ID and loop in a delay loop to allow the new master to initialize. A new master acquires the right to communicate lock and does any initialization it needs to, then releases it with its ID in the lock value block so others can again communicate.) Note that the naming master writes out the configuration file, while holding the right to communicate lock, when its local queries are done so that its config file is as complete as it can be made.
3. The naming master takes out (with a blocking AST) a lock which can be used to send it information, and another to send information back.
4. Once a node has the right to communicate lock, it reads its configuration file (which may not be the same as the naming master's config file). It first must check its system device, so uses the communication locks to send its system device information (id, port "letter" (which may be a Port Allocation Class), WW ID) to the name master for validation. If the naming master finds that this system device name is using a nonpreferred name, or has a different device name or port name than it thinks is permissible, it sends back a "nak" response to the sender, which then releases its locks and hangs. This prevents corrupting any disks, since to here the second node has only read data. Otherwise, the

second node does its own local disk scan and builds an in memory database of devices and path identification information. (This may involve some sysman connects.)

5. Now for each disk in the local database, the second system sends its information to the naming master and receives the naming master's preferred information (if any) for this device. This preferred information is used to replace the earlier local information where possible (or to hang the node if not). Information completely new to the naming master is added to the naming master's database also. Only information about local systems is sent here; names which will be served from elsewhere are to be known everywhere (so that a single configuration file might be widely shared) but not transmitted.
6. When the exchange from second to master system is over, the master system sends its information about other devices to the second system, which updates its in memory data about these if it differs.
7. This establishes the same global configuration data in memory. Now the secondary system and primary system take turns writing their configuration out to disk, while still holding the right to communicate lock. In this way any other systems will see the results of the negotiation.
8. This transaction continues as long as there are other systems which need to set their databases up. Once the transactions with the naming master are complete at a node, it goes on to set up the rest of its configuration, knowing that it will not cause any cluster name conflicts.

Ideally, all configuration files will be the same, but this system will build configuration files dynamically and cause them to converge in content even if they initially are mis-setup to be different. Where pairs of device names refer to the same storage, and where they cannot be discovered by any automatic means, they will have to be manually entered into the configuration database files. If this is done correctly, though, this method can be used to give stable naming.

7 Implementation:

To implement the chosen approach, some mods in INIT_IO_DB will be required to handle multi-letter port names (guess a month at most to do this). New locking code to run at the time SCSI class driver units are connected will need to be devised also. The configuration database will be read by the primitive file system, but need not be written until the full cluster boot is complete; it is expected that the name database will be kept in memory also for that long, if not for the life of the cluster. The amount of actual data needed is perhaps 4 quadwords per device, which remains tractable into quite huge configurations.

Existing SCSI naming may as well be retained for non - worldwide ID devices, either by using a different 2 letter device name prefix for the new devices, or by adding a constant allocation class to distinguish the new device names.

8 Risks:

The major difficulty is with boot devices, whose names can be checked for consistency only after the boot is well along. At that time if the wrong name has been chosen, all one can do is print a message and halt. This problem can be addressed with a SYSGEN parameter to hold the preferred device name (and perhaps the WW ID to go with it to prevent errors). There are precedents for doing this.

Configuration files will be in general created automatically and will need manual maintenance only where a system should not configure all devices it finds. This is currently rare, and should it come to be the rule later for, say, FC nets, it should suffice to provide a tool when there is need and perhaps to default in those cases that device names will be known but not used. The fact that we use a naming master in the architecture should segue neatly into using a name server on a FC net should that need arise also.

If console nicknames are to be supported, some means must be found to ensure they don't conflict. Conflicts would lead to possible unplanned name changes from one cluster boot to another, if the cluster sees a different console first in successive boots. The initial plan will be that console nicknames will be ignored in Phase I.