X3T9.2/91-160

to: Members, X3T9.2

fm: Steve Cornaby
Conner Peripherals

re: SCSI 3 Proposed Queueing Model

dt: 10 October 1991

After having reviewed the document, X3T9.2/91-098 Rev 1, a Proposed Queuing Model, I have some concerns about the ramifications of the current model.

In an attempt to clarify my concerns, I will first discuss the means by which an initiator would typically make file updates, then follow with methodologies which could make use of command queueing.

The types of initiators with which I am concerned are all multi-tasking, since this is the only environment that can currently make use of tagged command queueing. I will assume either a UNIX-based environment, or a file-server environment, since these are the most common.

UNIX is usually a heavily cached environment, which decreases redundant disk accesses and tends towards a bundling of disk writes. Files are typically updated based upon some time event. Upon this event, cached blocks which have been marked as "dirty" are written to the media, and file-system blocks are then updated to reflect the write.

File-servers, regardless of the particular implementation, funnel device requests to a central processing point. This central processor schedules disk accesses and alocates space. Most file-server software also includes some sort of scheduling algorithym to reduce actuator thrashing.

In current non-queued implementations, there is usually some sort of "pool" of disk requests which the O/S generates. The I/O sub-level operates on the pool to determine the order in which requests should be passed to the device. The O/S determines when the requests corresponding to a particular file node have been serviced and will then submit this update to the pool.

The pool concept works regardless of whether the system is single-user or multi-user, because the O/S generates the file system updates only after the associated writes have been completed.

Tagged command queueing can benefit the system to the extent that normal I/O sub-level performance techniques can now be off-loaded to the device. In addition, command overhead delays become less of a concern as they no longer cause a direct delay within the critical-time path. Additionally, command queueing allows distributed devices (e.g. arrays) to concurrently service multiple queued requests. Devices can now become multi-threading, with separate hardware servicing separate requests.

1/11

Current queued implementations with which I am familiar utilize only simple queue tags. The O/S retains the responsibility of holding off file node updates until the file has been correctly written.

There may be a case to be made for being able to give the device both the data updates and the file-system update at the same time, making sure that the file-system blocks are updated only upon successful completion of the affected data blocks. I think that I can see this intent behind the wording of the ORDERED QUEUE TAG message:

> If ORDERED QUEUE TAG messages are used, the target shall execute I/O Processes in the order received with respect to other I/O Processes received with ORDERED QUEUE TAG messages regardless of which initiator sends the I/O Processes. All I/O Processes received with a SIMPLE QUEUE TAG message prior to a I/O Process received with an ORDERED QUEUE TAG message, regardless of initiator, shall be executed before that I/O Process with the ORDERED QUEUE TAG message. All I/O Processes received with a SIMPLE QUEUE TAG message after a I/O Process received with an ORDERED QUEUE TAG message, regardless of initiator, shall be executed after that I/O Process with the ORDERED QUEUE TAG message.

Unfortunately, the solution has more of a DOS taste to it than the flavor of a multi-tasking system. The above scenario would work only in the case where a single file was being updated, and no multi-tasking was involved.

The above implementation quickly spells disaster in a multi-processing environment where multiple threads can be simultaneously outstanding. Suppose a system of 250 users, with a common thread loading of 10. Suppose further that file updates were made rather randomly, such that you could easily skew the file contents and the associated nodal updates. In this situation, with true multi-tasking, it can easily be understood how the intrusion of ORDERED QUEUE TAG commands would degrade disk performance by forcing far too frequent access to the file system areas of the device. Thrashing, not mechanical delay minimization is the end result.

If our intent is really to allow the system to give us all of the data requests followed by the file system requests, I suggest that the ORDERED QUEUE TAG is terribly inadequate for the task. What would instead be required is a thread concept that says, in simple terms:

> Do all of these simple queued tasks that BELONG TO THIS THREAD before doing this ordered queue task.

This extends the concept beyond the single-file update to the realm of multi-tasking.

I believe that George has done us all a great service in forcing us to model in what cases each of the queue type messages should be used. When the ORDERED QUEUE TAG was inserted into the document, I had no idea that it might be interpreted as is currently written into the model. I had interpred ordered queue tags as entirely independent from their simple counterparts, and had assumed that they would be used only for configuration or for error recovery. I do believe that this interpretation makes more sense than that found in the current model.

I suggest that wording that forces an interdependency between ordered queue tags and simple tags be deleted from the model.

If we do perceive a need in the world for being able to give the device both the blocks to be updated and the file system blocks to be updated at the same time, we should work to introduce a thread concept. This move would require that queue messages be modified somewhat. I might suggest the following:

> SIMPLE QUEUE TAG - (Same as current)
>
> HEAD OF QUEUE THREAD - (Defines first entry belonging to a thread)
>
> SIMPLE QUEUE THREAD - (Defines a subsequent entry belonging to a thread)
>
> ORDERED QUEUE THREAD - (Used for file system updates of the thread)

Each of the QUEUE THREAD messages would require an additional message byte as a thread tag which would uniquely define a thread.

I am also of the opinion that other utility messages would require definition to allow for more expedient cleanup. (Abort Thread, being one, to allow for error recovery of a threaded operation.) Current cleanup tools are inadequate.

As always, I appreciate the time and talents of the committe in reviewing these concept. I hope that by pooling our knowledge and talents, we can provide a framework for further I/O performance.