TECHNICAL RESEARCH REPORT

The Placement of File Sub-system Data Streams

by S. Gupta, J.S. Baras, N. Roussopoulos

CSHCN T.R. 96-14
(ISR T.R. 96-76)

The Center for Satellite and Hybrid Communication Networks is a NASA-sponsored Commercial Space Center also supported by the Department of Defense (DOD), industry, the State of Maryland, the University of Maryland and the Institute for Systems Research. This document is a technical report in the CSHCN series originating at the University of Maryland.

Web site http://www.isr.umd.edu/CSHCN/

The Placement of File Sub-system Data Streams *

Sandeep Gupta John S. Baras†

Nick Roussopoulos‡
Center for Satellite and Hybrid Communication Networks
Institute of Systems Research
University of Maryland, College Park MD

October 29, 1996

Abstract

An abstraction called Cues was implemented [1] for providing data streams over the file sub-system in the Unix kernel. The data stream of a Cue can be used for transfer of data as well as for control of the data stream by composing such structures. The algorithm for scheduling the compound Cue sequences read and write operations for data and control Cues in the kernel and retains other properties of user process driven read-write. This is possible for the data stream as this abstraction is over the file sub-system *and* the data stream is run in the context of the user process that created the Cue. This report reconsiders the architectural placement of the Cues, based on the experience with design and test of implementations. The underlying theme is to try to keep the control and flexibility as it is with user processes and keep from tying it in the kernel or user space unless specifically required, in effect re-evaluating the placement of the implementation in the kernel.

*This material is based upon work supported in part by the National Science Foundation under Grant No. NSF EEC 94-02384, and by the Center for Satellite and Hybrid Communication Networks under NASA contract NAGW-2777 and the State of Maryland.

†Also with Department of Electrical Engineering.

‡Also with Department of Computer Science, and Institute of Advanced Computer Studies.

1 Introduction

File sub-system data streams using Cues [1] are a data transfer service in the Unix kernel using the system abstractions provided to the *user* process. These are file descriptors, resource access permissions, and the user process context. They are all important to the design of these data streams, and in the scheduling of transfers for a Cue or with a pair of data and control Cues. In [1] measurements were reported on the improvement in performance of a disk to network transfer and in the saving of processor resources by eliminating data copy in and out of the kernel. Subsequently, for providing this as a service for moving large volumes of data such as Databases on a network, additional requirements for maintaining the access permissions were developed in [2]. All this is possible, because the design places the streams over the file sub-system and by limiting the scheduling of data transfers only during the time the process would normally run. The schedules were determined empirically for the machine used for tests cited [1], however the schedules require a very simple calculation based on the number of Cues and the time taken for the transfers. Keeping the user context allows the extension of this design for scheduling of these transfers with regard to the per-user resource utilization. The use of these abstractions makes porting Cue based streams to user space almost trivial. Two more extensions were considered desirable in the current design and this report reconsiders the architectural placement of Cues at the file sub-system in view of these extensions, the performance of existing systems, and empirical observations.

The idea behind the theme of trying to keep the abstraction free of user or kernel space restriction is to use the benefit of the experience gained with Cues to re-examine why and how this feature should be added to the kernel.

2 Background

In the rest of the report, the following familiarity with Cues, the system call `cue()` [1], and the terminology defined in [2] will be assumed: The system call takes two file descriptors as parameters and creates a data structure called Cue, one of which is the source, and the other the destination for data transfer. Such a Cue is identified by a new file descriptor returned by the system call and the descriptor is used to write control messages to direct the data transfer. In the basic case, the descriptors point to files or sockets, and the control messages are written to it by the process that owns the Cue. The control of a Cue created by a process may be granted to a peer process

connected to it via a socket, for a specified number and type of requests. This is done by providing the socket descriptor and a Cue descriptor to a second invocation of `cue()`. Such a Cue is called a Control Cue, and the former, a Data Cue. A Cue is said to be active if the process that owns it is in the kernel mode and it will perform a data transfer before returning to user mode. It is said to be running if it is performing a data transfer at such time. The data transfer path is called the horizontal path, and the control message path is called the vertical path. The control message may or may not have a parameter, and may be a Configuration or an Auxiliary message. Configuration messages are those that attempt to change the Cue configuration such as buffer size or transfer size in excess of what has been set by the process that owns them. The two design extensions desired are the presentation format for control parameters and ensuring robust operation of remote Cues in a general environment.

3 Presentation of Control Parameters

For using Cues on a distributed heterogeneous system, a presentation interface is required for mapping the control messages and their parameters from one architecture's native format to that of the other. The placement of service in the kernel entails that either these parameters be translated in the kernel, or the messages always be routed through the user process, where a presentation module or shared library be used. It is possible to do either or both, from the perspective of the feasibility and their use, with this design. The parameters with the Cue messages are integers and a large class of machines already include standard primitives for serializing and deserializing integers with the networking protocol implementations. Also, the parameters are an integer or a pair, so routing them through the user process will not be expensive. The protocol to control Cues is not complex and has a small number of messages. Should there be a need for an elaborate inter-Cue protocol such as that in case of two peer applications pacing control messages to each other, a mix of the two alternatives can be used easily. This is possible as the control Cue can forward the uninterpreted messages vertically to the user process. Even if the format conversion is coded in the kernel, this avoids hard coded design. Thus a protocol extension may be built over the existing set of messages without being limited by where the code for presentation of parameters is placed, without changing the design. From the perspective so far, this is modular, and the placement of Cues is compatible and interchangeable with user space placement.

4 Use of Control Streams for Remote Cues

The two basic messages required for a remote Cue are SET-OFFSET and TRANSFER-BYTES [1]. In the current implementation the two messages have integer codes, and these two have an integer parameter. Since the messages are this small, they can be, and must be encoded for serialization without much computation overhead. In the current tests of the implementation, fail-safe transport and identical presentation are guaranteed by the use of a reliable transport protocol on machines with similar architecture. It would be required to limit use of control Cues to the family of reliable transport sockets. On a connection-less control path there still is room for error due to lost messages. Connection-less sockets with certain protocols may also have to deal with interference from multiple sources. While these are familiar and solved problems, they begin to introduce underlying architecture specific limitations or modifications required of the system call. Though heterogeneity of data formats on different architecture is simple to solve, the robust operation of a Cue with remote control requires recovery from partially dropped messages, or those sent in error from the source. Since the two interacting processes communicating about the Cue are autonomous, the latter has to be taken into account. At the socket layer, there can be errors in messages received using datagram service, due to dropped messages, or due to out of order deliveries if a datagram protocol is used. At the sender there can be errors in framing the messages. A proper solution would require re-implementation of a reliable transport protocol, which amounts to duplication of function in the kernel. The protocol extension required will be small, as the number and time of interactions are limited, and it *can* be made independent of the placement in the user or the kernel space. Even so, adding support for connection-less streams begins to show the dependence on the lower layer protocol on the design. It does not add any new ability to the service, and disabling the use of connection-less sockets does not seem to take away anything. Error detection and handling is much simpler on a reliable transport, particularly because of in-order deliveries. Should the support for connection-less interface to remote Cues be required, a user space implementation will be able to handle the concomitant complexity due to the reasons outlined above.

From the design point of view, a reason Cues are implemented in kernel space is their performance due to elimination of copies and because of the use of scheduling in the kernel. The next two sections deal with the improvements in performance and the placement of Cue in the kernel, in view of the experience with the implementation and tests.

5 Sustaining versus Containing the Throughput

The objective of the initial implementation was to improve performance of the system by eliminating the extra copies from and back to the kernel. It also turned out that scheduling transfers in the kernel scaled better. In the tests reported in [1], transfers with multiple processes active at the same time tend to lose up to half of the system's bandwidth that would be achieved normally, were only one transfer active at that time. While this may be useful for the system configuration similar to the one in those tests, sometimes it may be useful to contain the throughput instead of sustaining it. One of the configurations of the system on which the call was ported, was capable of inundating the net to 60 % of its capacity on a single connection, with naive read-write sequences. As network interfaces and systems cannot always be guaranteed to be upgraded proportionately, there is use for both containing as well as sustaining the rate at which data is sent out of the system. The scheduling method for Cues is simple, and can be adapted to different rates, per connection. The kernel is the place where any such policy can be strictly enforced for naive applications, and the use for such a provision is not only limited to Cue streams, but can be used for other data transfers. This is not to say that user processes cannot pace the rate at which data is sent, and there is a flexibility trade-off if such enforcement is done in the kernel, leaving user processes without any option to use the entire bandwidth should they choose to use it at the configuration the system has been put together.

6 Elimination of Copies

During a transfer of data from one peripheral to another, the network and disk taken as examples in this work, there is a copy of data from the source device to the kernel, into user space, and back to the kernel and the destination device. Data that is not modified during such transfers uses more copies than are needed. The system call was one way of eliminating the copies. There can be other ways by either implementing the data stream using different abstractions [1], or by enhancing the calls that provide the standard data path to eliminate these copies. Bulk data movement is likely to remain an important activity on a lot of workstations. Some systems already provide support for minimizing copies during data transfer in hardware or by providing one such option on the file descriptors. Using either of these mechanisms if the data can be mapped into user space without copies

and vice versa then with respect to this feature it would be much easier to place Cues in user space. Use of kernel space and implementation is at the cost of adding additional complexity some of which may require handling of kernel specific nuances even though the test code for this call is fairly generic. User space implementations on the other hand can be relatively freely tested with the same abstraction, and in terms of robustness of the overall architecture, will be better in terms of this feature.

7 Summary

This report re-considered the placement of Cues. The current implementation was provided as a system call, and the data streams were placed over the file sub-system in the kernel thus providing the required user level abstractions. The experience with the implementation, the observations on the design extensions required, and the configuration of the newer machines prompted an evaluation of the advantages achieved using this placement. Most of the design seems to weigh out neutrally except when viewed from the perspective of newer host configurations. The dependence on the configuration of a host works *against* providing this service as a system call. Some newer features also make the option of implementing these streams as a user space application more attractive, while retaining the same abstraction.

References

[1] Sandeep Gupta, John S. Baras, Stephen Kelley, and Nick Roussopoulos. Cues: File subsystem data streams. Technical Report 96-53, Institute of Systems Research, Univesity of Maryland at College Park, 1996.

[2] Sandeep Gupta, John S. Baras, Stephen Kelley, and Nick Roussopoulos. Managing file subsystem data streams for databases on networked systems. Technical Report 96-60, Institute of Systems Research, Univesity of Maryland at College Park, 1996.