A Streaming Interface for Real-Time Interprocess Communication

Jork Löser Hermann Härtig Lars Reuther

Dresden University of Technology
Department of Computer Science
D-01062 Dresden, Germany
email: {jork, haertig, reuther}@os.inf.tu-dresden.de

Abstract

Timely transfer of long, continuous data streams and handling data omission are stringent requirements of multimedia applications. To cope with these requirements, we extend well known mechanisms for inter-address-space data transmission, such as zero-copy and fast IPC, by the notion of time. Therefore, we add a time track to data streams and add mechanisms to limit the validity of data. For cases of overload we add notification and revocation techniques.

1 Introduction

Efficient interprocess communication schemes for long data transfers in non real-time applications [DP93, MKT98, PDZ00] address the problem of copy avoidance by using shared memory. But they do not cover issues of time, such as data loss due to CPU shortage, what makes them inadequate for real-time and multimedia applications. Multimedia applications often consist of component chains processing a unidirectional stream of data in real-time. For reasons of efficient resource utilization, these systems are often designed for the average case instead of being designed for the worst case. As a result, resource shortage can happen and will lead to data loss then.

In this paper, we develop the requirements for a communication scheme, based on the constraints of a multimedia system with Quality-of-Service support. We describe the *DROPS Streaming Interface* (DSI) which is designed to support a time-triggered, shared-memory-based, producer-consumer-like communication between applications running on one node.

The remainder of the paper is organized as follows: Section 2 gives a more detailed overview of our Operating System DROPS and introduces into the current approaches for efficient interprocess communication. Section 3 derives the requirements to the communication system resulting from the demand for Quality-of-Service. Section 4 and 5 present the mechanisms for the DSI and some implementation details. In Section 6 performance results are presented, and Section 7 concludes.

2 Background

2.1 DROPS

DROPS is a system of cooperating resource managers that aims to provide QoS guarantees for applications. It is based on the L4-micro-kernel family [Lie95]. The main features of this micro-kernel family are protected address spaces, multiple threads per address space, and fast interprocess communication including operations to support memory sharing.

Resource managers are applications which manage all resources in the system. These include memory and CPU, but also complex resources based on others, such as the files of a file system. Typically, user applications connect multiple resource managers and establish so-called “resource chains”. Streams are used for communication in these resource chains. Figure 1 illustrates the application and the resource chain for the sender of a video-conferencing application.

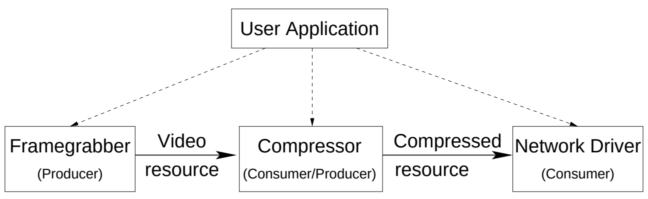

Figure 1: A chain of resources set up by an user application. The Frame grabber driver offers a video as a resource which is consumed by the compressor and converted to a compressed resource. This compressed resource is sent to the net.

DROPS optimizes its resource utilization by supporting mandatory and optional resources. Optional resources are requested with a probability, and are delivered if the underlying resources are available. This prioritizing of resources allows DROPS to support an imprecise computation model. Streams representing optional resources become optional too. They cope with a lack of resources by dropping parts of the stream.

In the event that the frame grabber card in Figure 1 produces the data with a higher frame rate than the compressor handles[1], after some time the buffer between the frame grabber and the compressor will be full. Now there are two options; either old or new data can be dropped. In our example it would be of limited use to discard the new data and to send obsolete data to the conferencing partner. Consequently, we drop the data of an old frame and allow the frame grabber to fill the buffers with new data as soon as it arrives.

In [Ham97] Hamann generalizes the idea of *jitter-constrained periodic streams*, a timely description for transported data between components. It is used to describe a stream of data packets of constant size by a certain bandwidth and a jitter. The jitter defines a maximum time between expected and real arrival of each packet. We use jitter-constrained periodic streams as a theoretical basis to achieve a quantitative description and management of the data.

2.2 Current Communication Interfaces

Various communication schemes have been published dealing with inter-address-space data transport. Unix sockets as the most common approach require copying of data in their original form. Druschel and Peterson introduced Fbufs [DP93], an efficient I/O buffer management facility to transfer data across protection-domain boundaries. Druschel extended his work, and together with Pai and Zwanenpoel he presented IO-Lite [PDZ00]. IO-Lite unifies all caching mechanisms in a system and allows them to use one single physical copy of the data. Pai et al demonstrated the influence of removal of multiple data buffering and the performance improvements gained by cross-subsystem optimization.

Albeit these systems are efficient, they lack a meaning of time and the missing resource control makes them ineligible for real-time transport. The video-conferencing application exemplifies this, none of the communication concepts we know of supports the dropping of data, once the communication system is in charge of.

[1]Frame grabbers typically have their own timers, a compressor may use another. This results in slight divergences of delivered and consumed data.

3 Requirements

The demand for Quality-of-Service in a system of communicating resource managers imposes a number of specific requirements to the used communication scheme. In this section, we derive these requirements.

Interaction and Isolation of Resource Managers: While in non real-time environments blocking communication concepts are sufficient, the different natures of applications in real-time environments require more complex synchronization schemes.

A resource representing a file read with a certain bandwidth from a disk should react to feedback from the consumer if the consumer reads data with a lower bandwidth than originally requested on resource creation. On the other hand, the video stream of the video-grabbing example in Section 2.1 can not simply be stopped. Hence, a communication scheme must support both the feedback to delay the data delivery of a resource, and the permanent supply of the communication peer with arriving data.

Putting aside the application context, resource managers additionally require isolation with respect to CPU usage at the communication channel. This is because of the optional resources which can fail to operate on one hand, and because of untrusting servers on the other hand. Resource managers must be able to continue their work, even if some of the resources they consume can not be delivered in time or some of their produced resources are not consumed.

Reservation of Resources: To guarantee availability of resources, real-time systems require to make reservation on limited and shared resources prior to their usage. This applies to the communication system as well. Basic resources needed for the data transfer of a stream must be reserved on stream creation.

Synchronization of Resources: While in non real-time environments blocking communication concepts are sufficient and automatically imply a synchronization, the different natures of applications in real-time environments require more complex synchronization schemes. As a consequence of making resources optional, data can be dropped in the middle of a stream. For a proper operation, the involved resource managers must be resynchronized. Thus, the communication scheme must provide synchronization information allowing the detection of and a proper reaction to losses.

Zero-copying: It is widely accepted and shown by multiple publications on inter-address-space communication interfaces that zero-copying is essential for a reasonable throughput between components [DP93, PDZ00, MKT98].

This is especially the case for micro-kernel-based systems where services are provided by multiple servers and the data path crosses address-space boundaries. As shown by Pai and Druschel in [PDZ00], mapping and unmapping of memory pages are costly and zero-copy implementations using dynamic mappings suffer from performance penalties. Thus, implementations using static mappings should be preferred to those reestablishing mappings during the data transfer.

Resource Sharing: One of the major aspects of DROPS is to provide optional high-level resources which share underlying low-level resources, e.g., physical memory is shared among multiple optional files. These low-level resources must sometimes be retracted, thus resource revocation must be supported for optional resources.

Support of non real-time Components: Some components do not need real-time behavior, but may want to communicate with resource managers. To allow these components to produce or consume data using the same interface, a non real-time mode must be available.

4 The DROPS Streaming Interface

The DROPS Streaming Interface (DSI) offers a producer-consumer communication interface for real-time applications. The produce and consume operations are modified to express timeliness, and also allow to specify that data is not available. The underlying transport mechanisms base on shared memory.

This section describes the mechanisms the DSI uses to fulfill the requirements listed in Section 3 in more detail.

4.1 Virtual time

DSI limits the validity of data by time, i.e. the data produced will expire after a certain time, even if it was not consumed. To do so, DSI uses virtual time, which is assigned to and stored together with each produced data. This virtual time corresponds to the position of the data in the entire stream. The mapping of virtual time to real-time is the responsibility of the consumer and the producer and must be defined on stream creation. By being visible to the consumer, this virtual time also provides the synchronization information required in Section 3.

There are applications, where the actuality of data is of second interest. When playing back a recorded video and resource shortage requires skipping of frames, it does not matter, which frame is skipped. Here, virtual time allows to “spool forward” and to skip a larger chunk of data. To do so, the receiver asks the producer to not to deliver more data until a certain virtual time, which dispenses the producer from some of its work. In the meantime, the receiver catches up with the old data in the buffers.

4.2 Shared-Memory Window

The QoS-constraints require to reserve all resources needed to perform the data transfer in advance. This applies to memory needed for buffering as well. DSI uses shared memory between the consumer and the producer for buffering and for the actual data transfer. DSI reserves this memory on stream creation.

In [Ham97] Hamann proved that it is necessary and sufficient to share a buffer with a certain size between the producer and the consumer to guarantee a requested Quality of Service. The data in this buffer belongs to a currently valid window of the potentially infinite stream. This closely resembles a consumer-producer problem using a shared ring buffer.

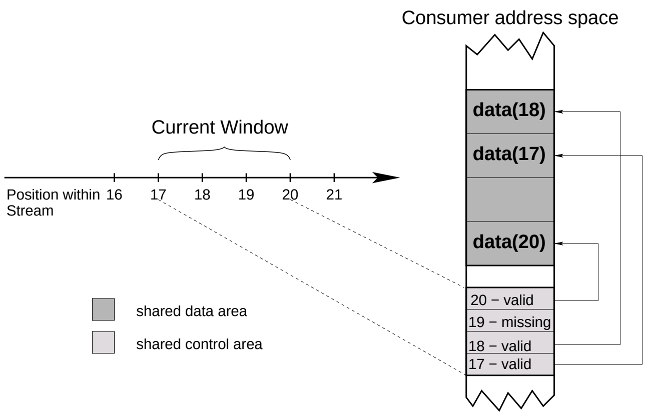

Figure 2: Window of valid data moving across the stream. While the descriptors at the control area are ordered, this is not necessary for the data area.

Some hardware supports scattered data buffers where data is not held in one contiguous chunk of memory. Instead, multiple descriptors give start address and size of each block. To cover this type of hardware without extra data copy operations, DSI provides support for these descriptors and establishes a second shared memory area, called the control area. This control area acts as an additional indirection to data access and consists of multiple packet descriptors. Figure 2 illustrates this indirection and the mapping of the stream window to the consumer address space. A produce operation includes writing data into the shared data area, writing the position and size of the data together with an associated virtual time into a packet descriptor at the control area and indicating the availability of new data in the packet. The consumer picks out the data and also indicates the successful read operation in the control area.

To support fast data forwarding between multiple resource managers, DSI offers memory chaining, i.e. consecutive resources in a resource chain can use the same data area. The control area used for indirection and synchronization is always assigned exclusively to one resource. An example is a network protocol manager. It could combine memory chaining with checksum offloading to prepend outgoing data packets with network headers without any additional copying overhead.

Summarising, DSI uses a consumer-producer scheme on a ring buffer containing packet descriptors. This allows DSI to easily manage the data area. By consuming packets, the consumer notifies the producer about which data has been consumed. Both the data area and the control area are reserved on stream creation, the data area can be shared between multiple streams.

4.3 Reclaiming of Memory Resources

To allow a resource sharing between multiple optional producers, DSI provides an operation to reclaim data that is produced but not consumed yet. To implement this, the mappings of the according data areas are established dynamically. This raises the problem on how the producer can notify the consumer of the reclaimed memory. A message sent to the consumer must not impose any blocking, but the producer must know for sure that the consumer will be notified prior to the next access to the memory. Hence, consumer and producer must agree in their contract about optional resources, that these resources can be withdrawn every time. The producer organizes the shared data region and retracts pages from the consumer with a “memory flush” operation. Pagefault signalling mechanisms are used to notify the consumer of the reclaimed memory[2].

If dynamically mapped memory is used in combination with memory chaining, requests reclaiming memory are passed on to all consumers of this memory along the resource chain.

DSI also uses the the memory reclaiming technique to enforce the temporal limitation of data. When the period of validity of a packet expires, the producer retracts the memory of the packet to get free buffers where it stores its new data. The consumer will be notified with a pagefault if it tries to access out-of-date data. As a reaction to this pagefault, the consumer can either use the next data in the shared buffer, or requests new data from the producer if the buffer is empty.

4.4 Blocking and Nonblocking Communication

To provide feedback about new data being sent and about data consumed at the receiver, DSI offers blocking communication. This means, the consumer is blocked if it wants to consume data and the buffer is empty. Corresponding to this, a producer is blocked if it wants to produce data, and the shared packet buffer is full.

Communication can also be done in a nonblocking fashion. Then, synchronization must rely on correct scheduling, the virtual time mechanism, and polling or signalling.

It is up to each resource manager to decide which mode of communication to use. In particular, this decision is independent of the mode of the communication peer.

[2] The operating system must support mechanisms to define functions handling pagefaults. The L4 μ-kernel provides this.

5 Implementation details

The synchronization mechanisms for blocking communication are implemented using the fast IPC-mechanisms of the L4 μ-kernel family. The communication scheme only involves the producer and the consumer, there is no third party needed acting as a trusted instance.

Synchronization requires blocking only on a consume operation if the buffer is empty, and on a produce operation if the buffer is full. In the other cases, no IPC is used. If both the producer and the consumer are well-synchronized, which is ensured by the scheduling mechanisms, the buffer is neither empty nor full. This means, in normal operation no IPCs are needed.

6 Performance Measurements

In this section, we present some performance measurements to demonstrate which costs the DSI design imposes on data transfer.

The measurement scenario consists of two resource managers transferring bulk data. The used packet sizes of 4 KB and greater are typical transfer sizes for multimedia data. We measured the time needed to produce a packet and to consume it. We measured four different modi of data transfer: the normal case using statically mapped shared memory with consumer and producer being well-synchronized; data transfer using statically mapped shared memory with blocking on consuming and producing; data transfer using dynamic page mapping and, for comparison reasons, data transfer based on copying[3]. The first mode corresponds to normal operation, and the second mode corresponds to the worst case occurring when consumer and producer are out of synchronization and need to block. The execution environment consisted of an Intel Pentium III with 750 MHz, running an L4 μ-kernel and the both resource managers, the results are the average of 1000 runs.

[3] The copying data transfer was implemented specifically for this measurement.

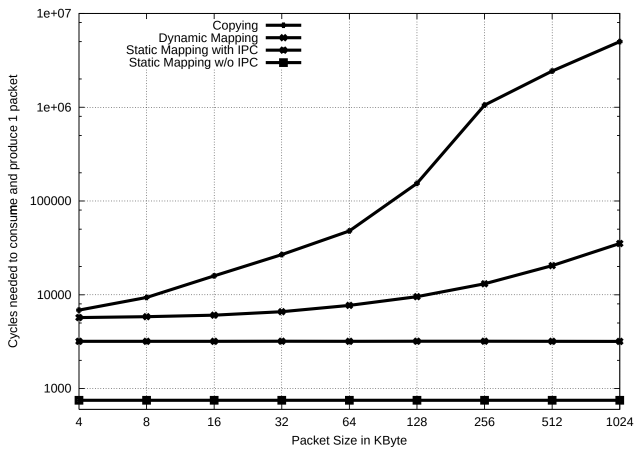

Figure 3: Time needed for transfer of 1 packet in CPU cycles.

The graph shows the moderate costs of about 750 CPU cycles for transferring a data packet with DSI when using statically mapped shared memory. When blocking occurs, the costs are higher because of the IPCs needed. Dynamic mapping imposes more overhead, linear to the size of the transferred packet. The copying results correlate to the memory throughput of the system, the inflexion point at 256KByte is based on cache influences.

7 Conclusion

In this paper we have shown which requirements must be fulfilled by a Quality-of-Service-aware communication scheme. We presented the DROPS Streaming Interface and illustrated that this interface is sufficient for real-time data communication by:

- limiting the validity of data by time,
- introducing a virtual time at the communication interface level,
- using a pre-allocated shared memory area of a limited size for communication,
- allowing to reclaim memory resources,
- and allowing blocking communication as well as non-blocking communication.

We have also shown, that the overhead that DSI imposes is reasonable, and efficient data transfer can be achieved.

References

[DP93] Peter Druschel and Larry L. Peterson. Fbufs: A high-bandwidth cross-domain transfer facility. In *14th ACM Symposium on Operating System Principles (SOSP)*, pages 189–202, Asheville, NC, December 1993.

[Ham97] Cl.-J. Hamann. On the quantitative specification of jitter constrained periodic streams. In *MASCOTS*, Haifa, Israel, January 1997.

[Lie95] J. Liedtke. On μ-kernel construction. In *15th ACM Symposium on Operating System Principles (SOSP)*, pages 237–250, Copper Mountain Resort, CO, December 1995.

[MKT98] Frank W. Miller, Pete Keleher, and Satish K. Tripathi. General data streaming. In *19th IEEE Real-Time Systems Sysmposium (RTSS)*, Madrid, Spain, December 1998.

[PDZ00] Vivek S. Pai, Peter Druschel, and Willy Zwanenpoel. IO-Lite: A unified I/O buffering and caching system. *ACM Transactions on Computer Systems*, 18(1):37–66, February 2000.