User-Mode DMA

Alberto Nava

Departamento de Computación

Universidad Simón Bolívar, Caracas, Venezuela

beto@plan9.cs.su.oz.au

Bob Kummerfeld

Basser Department of Computer Science

University of Sydney, Sydney, Australia

February 13, 1996

Abstract

In this article we explore user-mode DMA as a way to improve performance of user-mode file servers, reducing the temptation to keep them inside the kernel or as a special standalone programs. We describe our implementation and results of our experiments which show a 9% improvement over a conventional user-mode file server.

1 Introduction

During the last decade there was a tendency to remove functions from the operating system and migrate code to user mode servers [8, 2, 6, 7]. However, sometimes the performance degradation was too high or applications required so much from the

machine that user-mode servers were rejected. Servers were kept inside the kernel[2] or special standalone software was developed to handle such services[1, 6].

However, standalone software is difficult to create and modify, and sometimes different versions of the same components need to be developed and maintained. For example in Plan 9 [6] the file server kernel and the cpu kernel use different device drivers, which has been argued to be a problem[5].

It would be better to have all non-essential services in user-mode, and mechanisms to improve the performance of some of them. In this paper we explore user-mode DMA as a way to improve performance of user-mode file servers, and in particular multimedia file servers which perform large disk operations.

User-mode DMA eliminates copies of data between kernel and user space reducing delay and increasing bandwidth, and at the same time liberating the CPU for other tasks. The maximum reduction in delay will be the duration of the kernel to user-space copy, which depends on memory bandwidth and is usually not affected by the CPU cache.

We implemented user-mode DMA in Plan 9 and ran some tests which show that there is 4-9% performance improvement using it. These tests were done in a Home Area Multimedia File Server which we are developing as part of a project to build a Home Area Multimedia System[3].

The next section presents a performance comparison between a standalone and a user mode version of our file server. In Section 3 we discuss user-mode DMA and possible interfaces to the system. Our Plan 9 implementation and performance figures are given in Section 4 and 5. We finish with some conclusions in Section 6.

Clients	Mfs		Umfs	
	Bandwidth (Mbits/s)	Delay (ms)	Bandwidth (Mbits/s)	Delay (ms)
1	5.0	102	4.3	119
2	3.6	139	2.7	176

Table 1: Standalone vs. User Mode

2 User-Mode vs Standalone Comparison

To compare the performance of a user-mode file server versus a standalone one we did a series of experiments, in which we measured average bandwidth and delay. These experiments were done using Magnum 3000 machines connected by Ethernet. One machine was acting as file server and two others were doing sequential reads over big files, more than 20MB long. Disk layout and read-ahead strategy were the same in both file servers: 64KB blocks, a bitmap allocation structure and prefetching of the next block from disk.

Table 1 shows average bandwidth and delay for each file server with one and two clients. Bandwidth is the average over 10 sequential reads of the whole file, flushing the buffer cache each time. Delay represents the average time to read a block of a file. To consider the effect of *read-ahead* in the latency figure we measured it by reading the first 100 blocks of the file and taking the average.

¿From Table 1 we observe that **mfs**, the standalone file server, provides 16% more bandwidth than **umfs**, the user-mode file server, for one client, and 33% for two clients.

It's interesting to note that there is a $17ms$ difference in delay between **umfs** and **mfs** for one client. The possible causes for that are:

- Kernel entry/exit and context-switch overhead
- Extra movement of data
- Network overhead

Kernel entry/exit overhead will only account for a minor part of this $17ms$ overhead because kernel entry/exit is $20\mu s$ and we just have to get into the kernel 8 times to process a 64K read request. Moreover, there are not many context-switches because we only have one process running in user-mode.

Copying 64KB of memory from kernel space to user space takes $4.4ms$ in a Magnum 3000 machine, which could account for 25% of the extra time, and it is here where user-mode DMA will help us reduce delay and improve performance.

It iswell documented that the stream module in Plan 9 has some associated overhead[5], and at the same time the networking code in the standalone file server is very simple, and implements a minimal part of the protocol. We believe, but have not proved it yet, that the use of a minimalist networking code in the standalone server and the overhead of kernel streams account for a good fraction of this $17ms$.

3 User-Mode DMA

User-mode DMA will reduce movements of data between kernel and user space, which could reduce latency and improve the overall performance because the CPU will have more time to process other jobs or start other I/O operations.

The maximun reduction in delay would be the duration of the **memmove()** operation, which is usually limited by memory speed, not cache speed because most DMA buffers are flushed after the I/O operation or long sequential operations to user memory are not likely to be in cache. This is an important consideration because while CPU and network bandwidth are improving rapidly, memory bandwidth is not[4].

However, we will not necessarily regain all this time from the latency figure because seek and rotational delay could reduce the impact of user-mode DMA. For example, if we were reading the same block time after time, latency would not reduce much because the rotational delay will be much longer than the copy time, so even though we ask a few miliseconds before, the disk would have to wait for the data to come over the head. Nevertheless, CPU usage will reduce proportional to the copy time and to the number of active clients, which will improve the overall performance of the file server.

There are different interfaces to a user-mode DMA system depending on how much clients need to know in order to use it. We can group the interfaces in:

Zero-knowledge In a zero-knowledge system clients use standard I/O operations and allocation mechanisms. The system should discover when it is possible and appropriate to use user-memory for the I/O operation and not kernel buffers. Although it may sound the best approach, it is not always possible due to DMA memory restriction. For example, in some architectures, DMA memory needs to be in the first 8, 16 or 24M of memory, and aligned in 2^n boundaries, usually 64 bytes. Although the first constraint could be met by remapping the page to a lower frame in memory, if such a frame is available, the second is an unsolvable problem, which will limit the applicability of the user-mode DMA.

Complete-knowledge Users are responsible for providing buffers which are aligned for DMA. The kernel will check the integrity of such buffers and issue the appropiate I/O commands. This requires low level knowledge of the particular DMA system and will produce non-portable code.

DMA-compatible buffer pool An intermediate approach is to have a DMA compatible buffer pool, managed by the kernel. Whenever a client process wants to use user-mode DMA, it should first allocate a DMA compatible buffer from the

kernel and use it for later I/O operations. The kernel should implement such a pool, and check if one of those buffers is being used during I/O operations.

Because in our media file server the number of places where we could exploit user-mode DMA were limited and well known, we decided on a DMA-compatible pool solution which requires some special code in user-mode to handle DMA buffers, but which is easy to implement in the kernel and produces a portable solution.

4 Plan 9 Implementation

We implement user-mode DMA in Plan 9, using a new segment type called *dma*. Once a segment has been allocated (using **segattach**), we capture the first page-fault on it and allocate a kernel-buffer which will represent the real-memory for the virtual memory associated with the segment. This buffer must meet DMA constraints, which usually means being aligned in 2^n boundaries and being resident in memory. Later page-faults are resolved using the allocated buffer.

The new code consists of a new file server called *devumode*, and modifications to other memory related and SCSI related files:

devumode.c *Devumode* is a kernel-buffer allocator with a file server interface. It serves a data file in which we find information about segment availability and sizes, as well as memory mapping information. However, to allocated a new segment we use **segattach** with the new segment type *dma*. *Devumode* also implements **ualloc()**, and **ufree()** which are called by the page-fault handler whenever a page-fault occurs or a page is liberated in a segment of type *dma*.

devwren.c During SCSI reads and writes we check to see if the user's buffer corresponds with a previously allocated DMA kernel buffer, in which case we suppress the otherwise necessary **memmove()** operation.

Program	Latency per 64K (ms)	Max. Bandwidth (MBytes)
cat	23.14	2.8
ucat	21.16	3.0
theory	19.14	3.4

Table 2: User-Mode DMA, small benchmarks

5 Performance

To analyse the performance of the new feature we did several experiments. Some of then using small benchmarks and other using a user-mode file server handling local and remote connections. This file server was tuned for a small number of users, usually 2 or 3, so most of the tests we did were limited on the number of clients.

We used Magnum 3000 machines with 40MB memory bandwidth and 32K data cache. In this architecture moving 64Kb of memory takes $4.4ms$.

5.1 Small Test Programs

We measured the maximum bandwidth that we could obtain reading from a SCSI disk with and without user-mode DMA. For these tests we used Plan 9's low-level interface to the SCSI system, **/dev/sd0disk**.

Table 2 shows bandwidth and delay for three different situations: a normal **cat** on **/dev/sd0disk**, a modified version of **cat** which uses the new DMA feature, and theorical values based on a $4.4ms$ reduction in delay. We obtained a 9% bandwidth increase and a $1.84ms$ reduction in latency; values well below the maximum theorical values of, 20% and $4.4ms$.

Clients	dmafs	umfs	dmafs/umfs	mfs/dmafs
local	15.8	14.4	9%	
1 remote	4.5	4.3	4%	10%
2 remote	2.9	2.7	7%	24%

Table 3: User-Mode DMA based file server

The reason is that, on average, the disk has to rotate half way before the transfer phase started, which takes $8.3ms$ in this kind of disks. So even though we do reads $4.4ms$ ahead, the blocks are not ready $4.4ms$ before. We observed a lot of variance depending on how we access the disk (sequential, random or fixed) and whether the disk has an internal cache or not.

Nevertheless, user mode DMA provides 9% increased in bandwidth and also frees the CPU to do other activities as for example network processing, improving the overall performance.

5.2 User-Mode File Server

To evaluate the real effect of user-mode DMA we measured the bandwidth of a user mode file server using user-mode DMA and compared it with a normal user-mode file server and a standalone one.

Table 3 shows bandwidth for two different user-mode file servers and comparisons of their relative performance. FFrom this table we conclude that using user-mode DMA we obtained a 9% bandwidth improvement over a conventional user-mode file server for one local client and between 4-7% for one and two remote clients. However, we are still 10-24% behind the standalone one.

6 Conclusion

Even though during the last decade a lot of research was conducted in the area of micro-kernel based operating systems, high performance file servers are usually kept in the kernel or special standalone software is developed to obtain better performance. However, this software is difficult to develop and maintain, and software duplication is common, in particular device drivers and low-level software.

User-mode DMA is one of the options to reduce latency and improve the overall performance of user-level file servers. We implemented a variant of user-mode DMA, called *DMA-compatible buffer pool*, in which the kernel manages a pool of DMA compatible buffers which are allocated to user processes on demand.

We modified our user-mode file server to use the new feature and found a 9% improvement for local clients and 7% for remotes ones. Although we are still 15% behind the standalone file server, user-mode DMA is a step toward reducing the performance degradation of using user-mode file servers.

References

[1] Dave Hitz, James Lau, and Michael Malcolm. File System Design for an NFS File Server Appliance. In *USENIX Conference Proceedings*, pages 235–246, San Francisco, CA, Winter 1994. USENIX.

[2] Acceta M., Baron R. Golub D., Rashid R., Tevanian A., and Young M. Mach: A New Kernel for Unix Dvelopment. In *Proc. Summer 1986 USENIX Conf*, pages 93–112. USENIX, 1986.

[3] Alberto Nava and Bob Kummerfeld. Architecture of a Home Area Multimedia System. In *AUUG'95 and Asia-Pacific WWW'95 Conference*, pages 33–39, 1995.

[4] John K. Ousterhout. Why Aren't Operating Systems Getting Faster As Fast as Hardware? In *Proc. 1990 Summer USENIX Conf.*, Anaheim, June 11-15 1990.

[5] Rob Pike, Dave Presotto, Sean Dorward, Bob Flandrena, Ken Thompson, Howard Trickey, and Phil Winterbottom. Plan 9 from Bell Labs. In *Plan 9 - The Documents.* ATT, 1995.

[6] Rob Pike, Dave Presotto, Ken Thompson, and Howard Trickey. Plan 9 from Bell Labs. In *Summer 1990 UKUUG Conference*, number 17, pages 1–9, London, July 1990.

[7] M. Rozier, V. Abrossimov, F. Armand, I. Boule, M. Gien, M. Guillemont, F. Herrmann, C. Kaiser, S. Langlois, P. Leonard, and W. Neuhauser. Overview of the CHORUS Distributed Operating Systems. Technical Report CS-TR-90-25, Chorus Systems, 1990.

[8] Andrew S. Tanenbaum and Sape Mullender. An Overview of the Amoeba Distributed Operating System. *Operating Systems Review*, 15(3):51–64, July 1981.