Performance Analysis of Various Mechanisms for Inter-process Communication

Kwame Wright
Department of Electrical Engineering
The Cooper Union
wright2@cooper.edu

Kartik Gopalan   Hui Kang
Department of Computer Science
Binghamton University
{*kartik, hkang*}*@cs.binghamton.edu*

Abstract

Unix-based operating systems feature several forms of Inter-Process Communication (IPC) including pipes, Unix domain sockets, TCP/IP sockets, FIFOs, semaphores, and signals. Although these mechanisms provide similar services, each is designed differently with varying levels of performance. To better understand how these mechanisms work and why they perform the way they do, their implementations were analyzed. Pipes, Unix domain sockets, and TCP/IP sockets are commonly used mechanisms and are the focus of this project. Although they allow for communication between processes on the same machine, their performance in terms of throughput can vary significantly on the same platform.

The analysis of the IPC mechanisms involved examining the kernel source code of the Linux operating system and developing benchmark utilities to measure their performance. Overall, Unix domain sockets have proven to be the fastest mechanism, in many cases performing over two times faster than pipes and TCP/IP sockets. On the Intel Pentium III/4/Xeon processor systems used, pipes performed better than TCP/IP sockets over loopback, but the contrary was true for systems using Sun's UltraSPARC TI processor. This paper describes the tools developed and methods used to profile the IPC mechanisms.

1. Introduction

One important operating system responsibility is process management. In addition to process scheduling and resource allocation, an operating system must provide methods for separate processes to communicate with each other. These methods are known as Inter-Process Communication (IPC) mechanisms. IPC mechanisms allow for information sharing between processes and convenience for the user. In addition, they provide a way to modularize the operating system and to parallelize programs for computational speedup [1].

Prior research on improving IPC mechanisms has been done by Jochen Liedtke as described in his paper *Improving IPC by Kernel Design*. He realized the importance of IPC performance in modern operating systems and believed that unless they were fast, they would become unattractive for programmers, leading to inadequate use of threads and multitasking [2]. Liedtke's project focused on microkernels while this project focuses on the Linux kernel.

The idea for this project came from an interest in virtual machines. The concept of the virtual machine has been around since the 1960s. A virtual machine is an isolated execution environment within a computer allowing for multiple operating systems to be run simultaneously. Although hardware limitations have stunted the growth of virtual machines during the years following its conception, it is now more widely used due to the advances in computer technology. Processors are now fast enough, and memories are now large enough to allow for practical applications of virtual machines. Two examples of virtual machine monitors, or host software, are VMware and Xen [3].

The growing applications for virtual machines have led to the development of hardware optimizations. AMD and Intel, for example, both offer processor enhancements on some of their models to help improve the performance of virtual machines. Improvements are also being made on the software side in areas such as IPC. Imagine a single computer running two virtual machines. If one application running on one of the virtual machines wanted to exchange data with an application running on the other, it would have to do so via a software network using a standard network IPC mechanism such as a TCP socket. This is additional overhead as compared to the same two applications running on a single-domain machine.

Developments are in progress to design efficient IPC mechanisms specifically for communication between virtual machines. XenSockets and XWay are two examples. This project attempts to better understand how current IPC mechanisms work. The information gathered from this project will help improve existing mechanisms, and design new mechanisms that could be used directly between virtual machines.

2. Benchmarking Tools

2.1. Developed Software

Three utilities, `pipesend`, `udsocketsend`, and `tcpsocketsend` were developed to assist in benchmarking the three IPC mechanisms used. The three programs used pipes, Unix domain sockets, and TCP/IP sockets, respectively, to transmit a specified amount of data to a forked child process. They support up to four arguments to customize their functionality. These arguments are *bytes_to_send*, *buffer_size*, *repetitions*, and *logname_append*.

Bytes_to_send tells the program how much data to transmit, *buffer_size* the amount of data to send with each call of write, *repetitions* the number of times to repeat the transmission, and *logname_append* the string to append to the name of the log file generated. The output of each utility follows the same format. They return whether data was sent or received, the amount of data that was transferred, the time it took for the transfer to complete, the average rate at which the data was transferred, and the buffer size used.

2.2. OProfile

OProfile is a system-wide profiler for the Linux operating system. This software was used to profile the kernel while the three IPC mechanisms were being utilized, which allowed for a more in-depth analysis of the IPC mechanisms.

OProfile is freely available under the GNU GPL license. Version 0.9.2 of the profiler was used.

2.3. Test Hardware

Table 1 shows the 45 machines that were used to run the benchmarks.

Table 1: Hardware Tested

Hostname	RAM	CPU
alexander clausewitz drake eisenhower hannibal khan macarthur marshall montgomery nimitz patton rommel sun-tzu washington yamamoto	1GB	(2) Intel Xeon Dual-Core 2.00GHz
echo foxtrot golf hotel india juliet	256MB 527MB 384MB	Sun UltraSPARC IIi
krogoth maverick	8GB	(2) Intel Xeon Dual-Core 2.66GHz
brawn bumblebee grimlock inferno ironhide prowl ratchet slag snarl swoop wheeljack	512MB	Intel Pentium 4 2.53GHz
carrera charger countach gto mustang testarosa	1GB	Intel Pentium 4 2.26GHz
alpha bravo charlie delta	384MB 256MB	(2) Intel Pentium III
schroon*	3GB	AMD Athlon 64 X2 3800+

*Only used for OProfile

3. Performance Evaluation

3.1. Black-box Assessment

3.1.1. Methodology. To determine the relative performance of the IPC mechanisms, each was subjected to two benchmarks. These two benchmarks determined the raw throughput of each mechanism under different circumstances. The first benchmark involved transferring data ranging in size from 1 MB to 100 MBs with an equivalent buffer size so that there was only one system call being made to write. The second benchmark involved sending a fixed amount of data, 100 MBs, with varying buffer sizes from 100 KBs to 100 MBs.

There is a significant amount of overhead involved in making system calls due to the context switching [4]. The second benchmark was designed to get a better understanding of how the various combinations of memory operations and context switching affected performance. Both benchmarks were performed on many machines with varying hardware configurations to not only minimize the effects of the differences in hardware, but to also understand what effect the hardware had on these mechanisms.

3.1.2. Observations. It was hypothesized that pipes would have the highest throughtput due to its limited functionality, since it is half-duplex, but this was not true. For almost all of the data sizes transferred, Unix domain sockets performed better than both TCP sockets and pipes, as can be seen in Figure 1 below. Figure 1 shows the transfer rates for the IPC mechanisms, but it should be noted that they do not represent the speeds obtained by all of the test machines. The transfer rates are consistent across the machines with similar hardware configurations though. On some machines, Unix domain sockets reached transfer rates as high as 1500 MB/s.

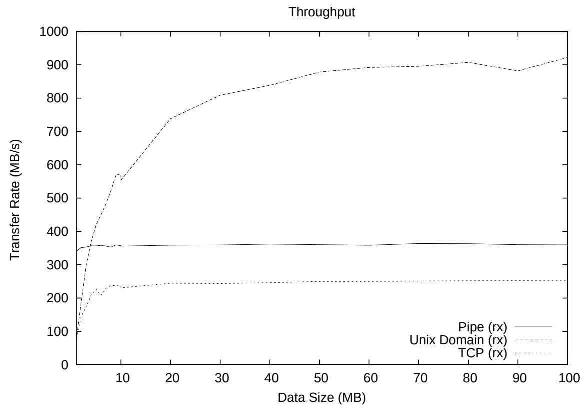

Figure 1: Benchmark 1 on bumblebee

For small data sizes, the throughput of Unix domain sockets was below that of pipes. Investigation of the cause was inconclusive because the results were not consistent across all machines. On the Intel processor machines, pipes performed better than TCP sockets (see Figure 1). This was not true for the machines using the UltraSPARC processor. This is probably due to the UltraSPARC system's design for network-intensive applications. It was not expected, but at the same time it is not entirely surprising that this was the case. There will be further research to determine what caused this to happen.

Figure 2 shows the results of the second benchmark. While the performance of pipes and TCP sockets remained relatively stable, Unix domain sockets showed a major decrease in performance as the buffer size increased.

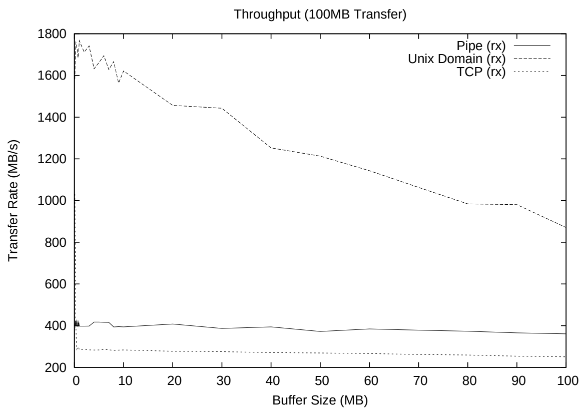

Figure 2: Benchmark 2 on bumblebee

This is an interesting result because as the buffer size is increased, the number of write system calls decreases, meaning there is less context switching overhead involved. Unix domain sockets exhibited about double the throughput when it dealt with small pieces of data at a time, but further investigation will be needed before any conclusions can be drawn.

3.2. Gray-box Assessment

3.2.1. Methodology. For a few data sizes ranging from 1 KB to 100 MBs, the benchmark tools were monitored using OProfile which allowed for the system call usage to be monitored for each program. Using the merge parameter of the `opreport` tool included with OProfile, the events could be separated by the individual applications, process ids, or CPUs allowing for an even better understanding of what was taking place as the IPC mechanisms were being used. Since the benchmarking tools used a forked child to send/receive data, separating the reports by process id allowed for the isolation of the sender from the receiver. The profiling utility was set to take samples on CPU_CLK_UNHALTED events, or cycles out of halt state, every three-thousand counts. Figure 3 shows histograms for the reports from the 1KB and 100MB transfers for comparison. Note that the symbols are listed in order of increasing percentage, with the smallest at the top.

3.2.2. Observations. At about 1 MB for all of the mechanisms, memory copies from user-space to

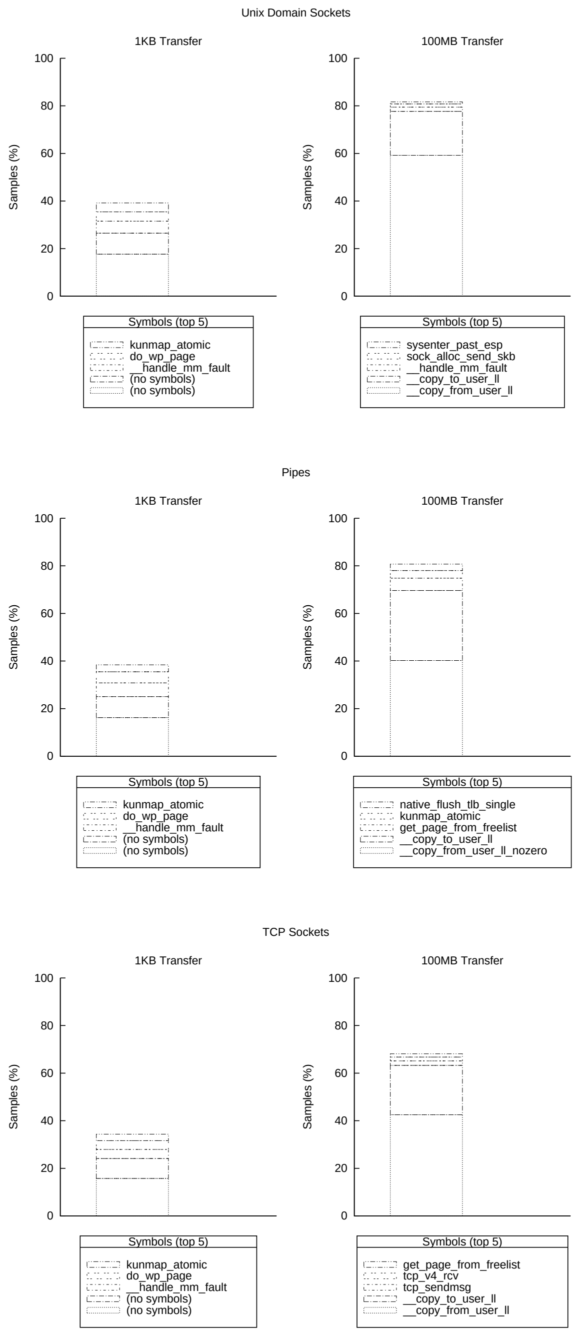

Figure 3: Histograms of OProfile reports

kernel-space, and vice versa, became the dominating symbols, or functions, taking up at least 50% of all the events detected by OProfile. This percentage increases for all of the mechanisms as the transfer sizes become larger, but it does so at different rates and the mechanisms continue to exhibit performance differences up to 100 MB. The '(no symbols)' shown on the histograms are associated with libc-2.5.so and ld-2.5.so. OProfile was unable to report on them because they were not compiled with debugging options, but this problem will be revised.

4. Conclusion

Unix domain sockets have proven to deliver the highest throughput when compared to the other mechanisms. While its dominance is still unclear for transfers of small amounts of data, it is otherwise the best mechanism to use within a single machine.

Now that the IPC mechanisms have been analyzed, the next step is to attempt to develop performance optimizations for them. This will involve hands-on work with the kernel source code. Significant results in this area are not guaranteed but it will provide a better understanding of IPCs which can be applied to developing new mechanisms for virtual machines, or enhancing existing ones.

Acknowledgments

Thanks goes to Michael Lewis for proofreading this paper and providing helpful suggestions. This paper was written using $\LaTeX$.

References

[1] Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne. *Operating System Concepts, 7th edition*. John Wiley & Sons, Hoboken, NJ, 2005.

[2] Jochen Liedtke. Improving IPC by kernel design. In *SOSP '93: Proceedings of the fourteenth ACM symposium on Operating systems principles*, pages 175–188, New York, NY, USA, 1993. ACM Press.

[3] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM Press.

[4] Brian N. Bershad, Thomas E. Anderson, Edward D. Lazowska, and Henry M. Levy. User-level interprocess communication for shared memory multiprocessors. *ACM Trans. Comput. Syst.*, 9(2):175–198, 1991.

[5] W. Richard Stevens and Stephen A. Rago. *Advanced Programming in the Unix Environment*. Addison-Wesley, Westford, Massachusetts, 1992.

[6] W. Richard Stevens, Bill Fenner, and Andrew M. Rudoff. *Unix Network Programming: The Sockets Networking API, Volume 1 - Third Edition*. Addison-Wesley, Westford, Massachusetts, 2004.