Abstract: μ-kernel meets Unix Pipes

Hermann Härtig Michael Hohmuth Sebastian Schönberg Jean Wolter

June 9, 1997

1 Pipes

It is widely accepted that IPC can be implemented significantly faster in a μ-kernel environment than in classical monolithic systems. However, applications have to be rewritten to make use of it. In this abstract we study various degrees of emulations of classical IPC on μ-kernel IPC to get an estimation for the cost of emulation on various levels.

1.1 The Experiment

We compared seven different implementations on a single platform[1]:

1. **Pipe on monolithic Linux**
   We used the code provided by the `lmbench` [McVoy and Saetlin 1996] package to obtain performance results for comparison.

2. **Pipe on our Linux Port, ABI emulated**
   The same binary code that was used on (1), but run on the L4-based Linux version.

3. **Pipe on our Linux Port, using a shared C library which avoids the trampoline cost (non ABI emulation)**
   Since binary emulation results in a trampoline effect due to exception handling of the `syscall` [Intel Corporation 1995] instruction, the non-ABI emulation avoids this overhead using L4 IPC to directly communicate with the Linux server task.

4. **Pipe implementation using L4 IPC**
   A fully functional asynchronous pipe implementation but on user level using L4 IPC. The Linux server task is not involved in the operation. Due to the highly optimized message transfer operation of the L4 IPC and a smaller amount of kernel crossing calls, this implementation was expected to get the best performance results while keeping the same functionality. The structure of this experiment is shown in figure 1.

5. **Native synchronous L4 IPC**
   In opposite to (4), the asynchronity of pipes has been abandoned. The difference in bandwidth and latency are the costs to be payed for asynchronity.

6. **Pipe on Mk-Linux with in-kernel Linux**

7. **Pipe on Mk-Linux with user-level Linux**
   These last two experiments have been performed to gain comparable results of the Mach-based implementation of Linux [des Places et al. 1996]. Trusted Mach servers can be loaded "in-kernel". That reduces IPC costs and additional costs for address space switches.

The latency was measured using "ping-pong" of 1 byte buffers, bandwidth using ping-pong of 64KB buffers. The measurement programs are written in C. Since C libraries use very large chunks of address spaces, IPC was rather expensive. Significantly better results can be achieved on architectures with lower hardware-induced system-call overhead.

These results show that the latency of original monolithic pipe implementation can almost be achieved by emulating asynchronous pipe operations on synchronous L4 IPC. The bandwidth can be increased. Using strictly synchronous L4 IPC requires changes to some applications but delivers better results.

[1] 133MHz Pentium, 64MB asynch. RAM

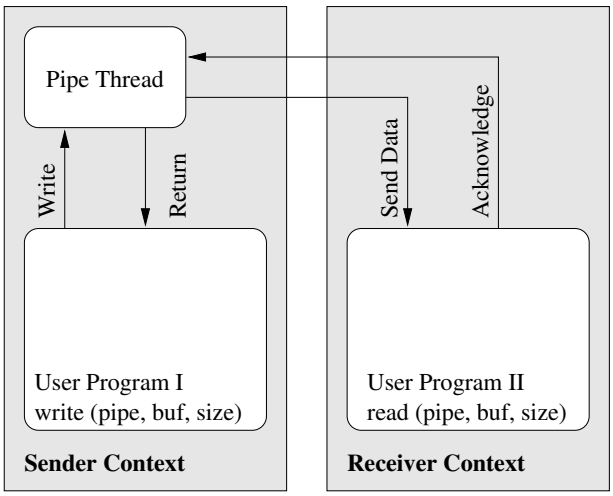

Figure 1: Asynchronous Pipe and L4

System	Latency	Throughput
Linux monolithic	29 μs	41 MB/s
Linux L4, ABI emul	56 μs	38 MB/s
Linux L4, n-ABI	46 μs	40 MB/s
L4/IPC pipe	32 μs	70 MB/s
L4 native IPC	10 μs	105 MB/s
Mk-Linux in-kernel	230 μs	9 MB/s
Mk-Linux user	850 μs	5 MB/s

Table 1: Pipe I/O Performance

The achieved throughput, listed in table 1, is shown in figure 2.

1.2 Conclusion

Our experiments have shown that the usage of "native μ-kernel" constructs like IPC allow a significant performance improvement of some standard Unix functions without changing the functionality.

References

des Places, F. B., Stephen, N., and Reynolds, F. D. 1996. Linux on the OSF Mach3 microkernel. In *Conference on Freely Distributable Software*.

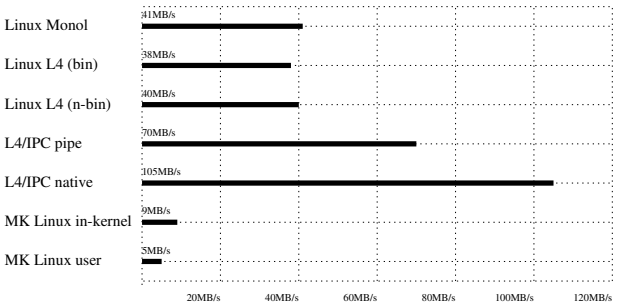

Figure 2: Message Throughput

Intel Corporation 1995. *Pentium User Reference Manual*. St. Clara: Intel Corporation.

McVoy, L. and Saetlin, C. 1996. lmbench: Portable tools for performance analysis. Tech. rep., Silicon Graphics, Inc., Hewlett-Packard Laboratories.