Towards decentralized memory management in a multikernel

Simon Gerber*, Timothy Roscoe
{simon.gerber,troscoe}@inf.ethz.ch

Sep 9, 2012

Abstract

Seeing that modern hardware has more and more cores, we feel that classic, centralized memory management is not the ideal solution, especially in conjunction with a multikernel operating system [1].

While there are solutions that address some of the limitations of a centralized memory system in a multicore environment [2, 3], we feel that we can do better by designing a new approach that is based on a share-nothing architecture, explicitly handles distributedness and non-uniformness, and makes use of available hardware features to provide sharing when the user requests it.

Additionally, we want to be able to make strong guarantees about the security of private memory regions and about the distribution of physical memory to applications [5]. To do this, we enlist the help of a strict resource management system (e.g. capabilities [6, 7]) that gives its own guarantees about who gets to access a particular resource (e.g. a region of physical memory).

The first step in order to make such a design work is to make the resource manager (which is in charge of access permissions) and the page table manipulation mechanisms aware of each other so that at any given time, given either a page table entry or a resource handle (capability) we know who has access / where the resource indicated by the handle is mapped [4].

First experiments using a basic implementation of that idea show that keeping in-kernel state for each mapped memory region has a relatively low overhead that is offset by the fact that the new implementation batches updates to sequential page table entries for the same mapping for each leaf page table.

Having these basic operations, we can then start to experiment with ideas for a decentralized and agile memory management system. In a first step we are designing a decentralized memory management system for homogenous systems that is agile in adapting itself to the ever-changing multicore hardware. This design will serve to show if our idea is sound and we can then extend the design to take into account heterogeneity.

One way to extend our memory management system to be able to handle hetergenous systems (e.g. mixed IA-32 and x86-64 cores) is to create a sufficiently abstract page table format that – when replicated on each core – can be translated by the cores into their local hardware page table format. This will enable us to investigate policies of where to construct page tables for which cores regardless of the exact hardware page table format by having a "template" architecture-agnostic page table layout.

Another area to explore presents itself when looking at sharing as a local optimization of replication. If we want to treat sharing of memory objects as an optimization for replication, we need to have support for fine-grained shared-memory regions in the memory managmement system.

An open question is how to handle shared libraries in the presence of a operating system that is fundamentally share-nothing. One way this could be implemented is by actually replicating the shared library on all cores and modifying each core-local page table tree to include the pages where the local copy of the shared library is kept.

*student

References

[1] A. Baumann, P. Barham, P.-E. Dagand, T. Harris, R. Isaacs, S. Peter, T. Roscoe, A. Schüpbach, and A. Singhania. The multikernel: a new OS architecture for scalable multicore systems. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, SOSP '09, pages 29–44, New York, NY, USA, 2009. ACM.

[2] S. Boyd-Wickizer, H. Chen, R. Chen, Y. Mao, F. Kaashoek, R. Morris, A. Pesterev, L. Stein, M. Wu, Y. Dai, Y. Zhang, and Z. Zhang. Corey: an operating system for many cores. In *Proceedings of the 8th USENIX conference on Operating systems design and implementation*, OSDI'08, pages 43–57, Berkeley, CA, USA, 2008. USENIX Association.

[3] B. Gamsa, O. Krieger, J. Appavoo, and M. Stumm. Tornado: maximizing locality and concurrency in a shared memory multiprocessor operating system. In *Proceedings of the third symposium on Operating systems design and implementation*, OSDI '99, pages 87–100, Berkeley, CA, USA, 1999. USENIX Association.

[4] S. Gerber. Virtual memory in a multikernel. Master's thesis, ETH Zürich, May 2012.

[5] S. M. Hand. Self-paging in the Nemesis operating system. In *Proceedings of the third symposium on Operating systems design and implementation*, OSDI '99, pages 73–86, Berkeley, CA, USA, 1999. USENIX Association.

[6] N. Hardy. KeyKOS architecture. *SIGOPS Oper. Syst. Rev.*, 19(4):8–25, Oct. 1985.

[7] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood. seL4: formal verification of an OS kernel. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, SOSP '09, pages 207–220, New York, NY, USA, 2009. ACM.

Towards decentralized memory management in a multikernel

Simon Gerber, Timothy Roscoe / Systems Group, ETH Zürich

www.barrelfish.org

[1] Problem Statement

- Current OSes try to adapt memory-management to heterogeneous and multicore systems
- Problem: high synchronization overhead and contention on shared data structures
- Solution: Apply multikernel design principles to memory management

[2] Current Solution: Adapt Existing Systems

Try to make memory management scale by:

- Fine-grained locking of shared data structures (page tables)
- Synchronizing changes across cores
- Adapting software to hardware cache-coherence protocols

Changing one subsystem influences a lot of others

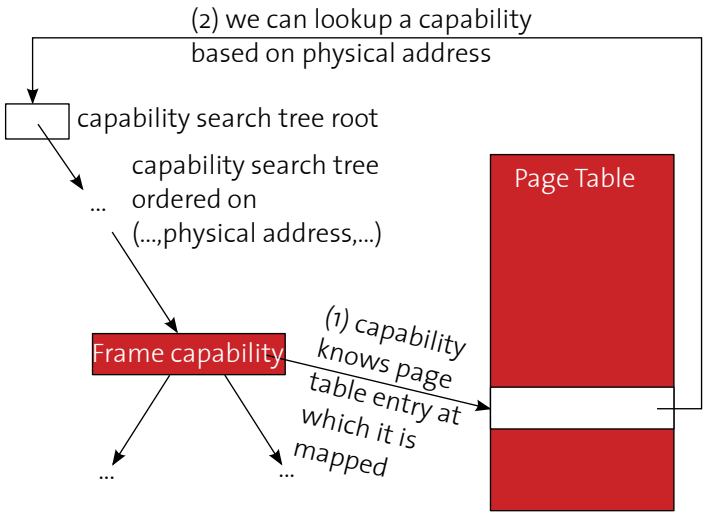
[3] Our Approach: Apply Multikernel Principles to MM

Build a memory management system using:

- **Explicit communication** using distributed systems techniques
- **Share-nothing architecture**
- Treating caches and main memory as distinct units

[3ext] Our Approach: Implementation Details

- First prototype built in the Barrelfish research operating system
- Make resource management (capabilities) and memory management aware of each other



- Separate mechanisms and policy: Utilize **self-paging**
- Default policy: **replicate** all page tables and associated capabilities
- Security: The resource management system checks access rights when installing mappings

[4] Conclusion

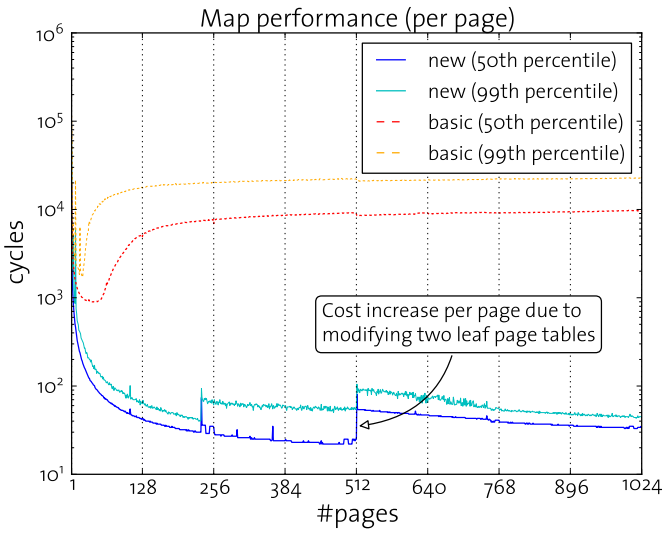
- Early implementation looks promising
- In-kernel bookkeeping does not need to have big impact on performance.

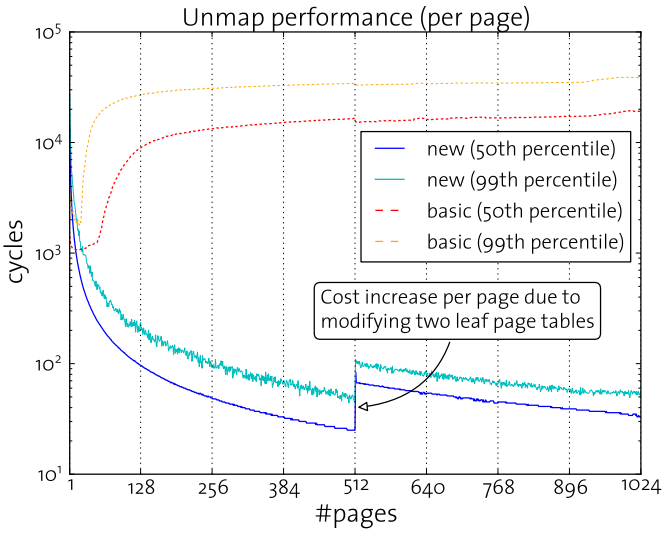
[5] Future Plans and Open Questions

- Add support for heterogeneous systems
- Create fine-grained memory-sharing mechanisms
- Enable shared library usage

[6] Early Results

Performance improvement for new design: batch updates for sequential entries in the same leaf page table.





ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich
inf Informatik Computer Science
Systems@ETH Zürich