Mach Virtual Memory

Jehan-François Pâris

Computer Science Department
University of Houston
Houston, TX 77204-3475

1. General Objectives

Mach's virtual memory system has two primary objectives: (a) to be as portable as the UNIX virtual memory system while supporting more functionality (see below), and (b) to support multiprocessing, distributed systems and large address spaces.

2. The Virtual Memory User Interface

Its main features are:

1. *A consistent virtual memory interface on all machines supporting Mach*: some features, such as shared pages, can be more or less efficiently implemented depending on the underlying hardware;
2. *Full support for multiprocessing*: this includes *thread* support, efficient data sharing mechanisms and a fully parallel implementation of the virtual memory;
3. *Modular paging*: there is no dedicated swap area and external pagers are allowed to implement file mapping or any application-specific paging policy (such as recoverable virtual memory for transaction management).

3. Data Structures and Algorithms

There are four basic data structures:

1. The *resident page table*: keeps track of Mach pages residing in main memory (a Mach page may contain several physical pages but their number must be a power of 2);
2. The *memory object*: a unit of backing storage managed by the kernel or a user task[1];
3. The *address map*: a doubly linked list of map entries each which describes a mapping from a range of virtual addresses to a region of a memory object; all pages mapped by the same map entry have the same protection and inheritance attributes (page is to be copied, shared or ignored at **`fork()`** time) .
4. The *p-map*: the memory-mapping data structure used by the specific hardware (page tables for a VAX, inverted page table for the IBM RT, more complex structures on other machines); it does not have to be kept fully up-to-date except for the pages in the resident part of the kernel.

[1] In practice, this means either a file or a swap area.

The address map is a list of chunks of page tables. "Hint pointers" are kept to shorten searches besides most address maps are expected to contain few entries.

Mach systematically uses lazy evaluation: invalid page faults are used to postpone the creation of page tables and lookup of disk addresses until needed. For instance, Mach implements stacks for UNIX processes by allocating enough virtual memory to hold the maximum size allowed for each process but delays the actual mapping until the page is referenced. Hardware-detected protection violations are also used to implement *copy-on-write*.

Mach uses a *global FI*FO memory policy but places expelled pages on an inactive list from which they can be reclaimed. This the same policy as VMS but for the fact that VMS allocated a separate resident set of frames to each process.

All virtual memory algorithms rely on *locks* whenever exclusive access to a data structure has to be guaranteed. To prevent *deadlocks*, all algorithms gain locks using the same ordering.

The total size of the machine-dependent part of Mach's virtual memory implementation is about 16 kilobytes.

4. Applications

Mach uses its memory object mapping mechanism to implement standard UNIX I/O semanticswhile allowing user programs to access directly the mapped file data.

As in Accent [1], *copy-on-write* is used to implement efficient message passing: messages can be sent and received without having any data copied until either the sender or the receiver tries to modify the data. It also provides a much faster implementation of the UNIX **`fork()`** and eliminates the **`vfork()`** kludge.

Shared libraries[2] are supported through the mapped file interface.

References

[1] Robert Fitzgerald and Richard F. Rashid, "The Integration of Virtual Memory Management and Interprocess Communication in Accent." *ACM Trans. on Computer Systems*, 4, 2 (1986), pp. 147-177.

[2] R. Rashid, A. Tevanian, M. Young, D. Golub, R. Baron, D. Black, W. Bolosky and J. Chew, "Machine-independent virtual memory management for paged uniprocessor and multiprocessor architectures," *IEEE Trans. on Computers*, C-37:8 (1988), pp. 896-905.

[3] A. Tevanian Jr., A*rchitecture-Independent Virtual Memory Management for Parallel and Distributed Environments: The Mach Approach*. Ph.D. Dissertation, Technical Report CMU-CS-88-106, Computer Science Department, Carnegie Mellon University (1987).

[2] A *shared library* is a system library that is dynamically shared among all processes currently in main memory.