Instructor Notes

Mach

Machine-Independent Virtual Memory Management for Paged Uniprocessor and Multiprocessor Architectures, by *R. Rashid and A. Tevanian and M. Young and D. Golub and R. Baron and D. Black and W. Bolosky and J. Chew,*
Proceedings of the 2nd International Conference on Architectural Support for Programming Languages and Operating System (ASPLOS), 1987.

1. What problem did Mach address?

Traditonal UNIX: had to rewrite large portions of VM system for each platform

2. At a high-level, what is their solution?

Separate machine independent from machine dependent components

Keep machine dependent components (the parts that have to change) as simple as possible (treat as a cache in many cases)

Use proper data structures to help with mapping:

Goal: Support v.a. -> memory object -> p.a. mapping efficiently

3. What 5 basic abstractions did Mach rely upon? What needs to be implemented efficiently for an extensible system?

    a. task: process – v.a. space, unit of protection for system resources

    b. thread: unit of CPU utilization (PC)

    c. port: communication channel; logical queue for messages; protected by kernel; send and receive.

    d. Message: typed collection of data
        a. Adv: helps with extensible systems and distributed systems
        b. Requirements: efficient copy-on-write for local environments

    e. Memory object: collection of data managed by a server, mapped into address space of task

4. What functionality is provided to manage an address space?

    a. Allocate/deallocate virtual memory (grow or shrink) fill w/zeros or copy

    b. Specify inheritance for child tasks (shared, copy, or none) – page level
        i. When create child, defines how a.s. is initialized
        ii. What would UNIX for() specify? – copy
            1. How to make efficient? Use copy-on-write for address map(default)

    c. Set protection of region – page level
        i. r/w/x – enforcement requires hardware support
        ii. current: level enforced by hardware
        iii. maximum: can't be raised by process, current <= maximum
        iv. mask away rights for safety

    d. Pagein and pagout (read and write) – manage as a pager

5. What 4 primary data structures are used for memory management in Mach?

Resident page table: tracks info about machine independent pages

Addrss map: doubly-linked list of map entries, describes mapping from range of va to region of memory object

Memory object: unit of backing storage managed by kernel or user task

Pmap: machine dependent mapping data structure (hardware defined!)

6. What is the purpose of the **resident page table**? How is it organized? What info is tracked for each page?

    a. Tells us state of each (machine independent page) of physical memory
    b. Table indexed by physical page number:
        i. Modified (to write back), reference bits (replacement policy)

7. Page entries from the resident page table may simultaneously be linked in 3 different lists. What are the 3 lists and what are their purposes?

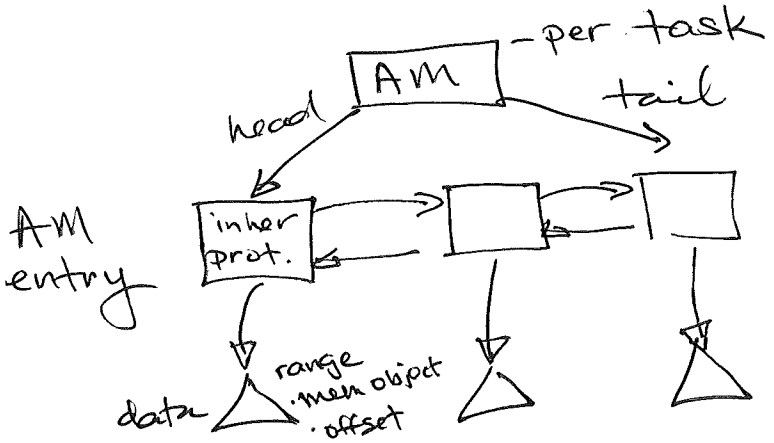
    a. Memory object list: all pages within object
        i. Why? Simplify deallocation and copying of object

    b. Memory allocation queues: free, reclaimable, and allocated pages for paging daemon (help find appropriate page)

    c. Object/offset hash bucket: Why?
        i. Get pa fast on "page fault"
        ii. How else could find? Search through memory object list
        iii. Why must resident table handle page fault?
            1. Page could still be in physical memory
            2. HW (pmap) doesn't know all translations

8. What is the purpose of the **address map** per task address space? How is it organized? What are the advantages of this structure? How many entries does it typically contain?

    a. Purpose: maps va -> byte offsets in memory objects
    b. Doubly-linked list of address map entries, each of which maps continguous ranges (sorted), all have same inherieance and protection attributes
    c. How many entries typically?
        i. Code, stack, heap (init and uninit)
    d. Advantages?
        i. Small
        ii. Does not penalize large, sparse address spaces
        iii. Quick to go from v.a. to memory object
    e. When accessed?
        Page fault lookup (If must get data from memory object itself)
        Copy/protection changes
        Alloc/dealloc address ranges



9. What is the purpose of **memory objects?** What is the purpose of the reference counter for each memory object? What handles page faults for each memory object?

    a. Repository for data, indexed by byte (backing store); read or write on it
    b. Ref count: free when no more mapped references to it; keep popular ones around after gone; What is this?
        i. File cache
    c. What handles page faults for each memory object?
        i. Pager; identified by pagin-object port

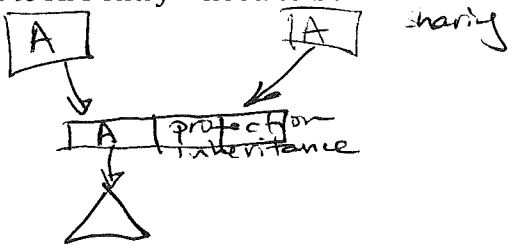
10. Why is efficient copy-on-write needed in Mach? How does copy-on-write work? What inefficiencies can their approach cause? How does read/write sharing work?

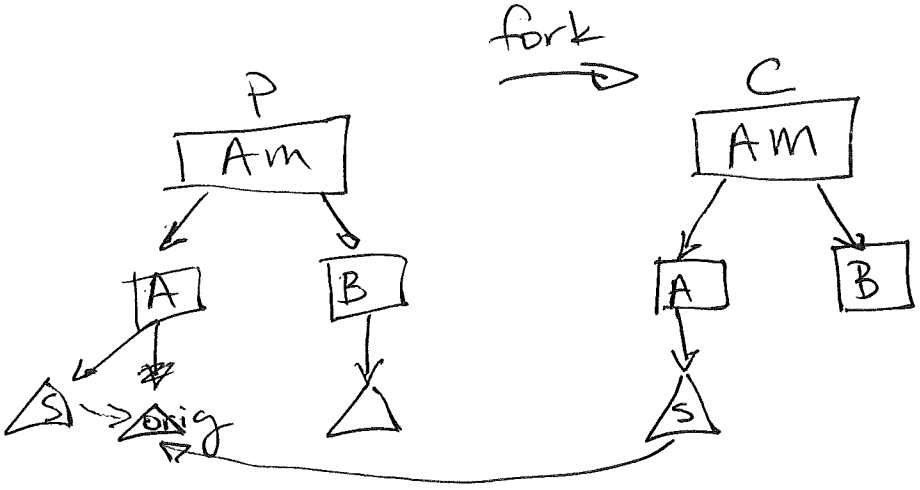
Mach performs lots of IPC – microkernel for multiprocessors.

Copy-on-write lets messages be shared efficiently

Use shadow objects which are 2nd-level in address map to point to memory object (can remove when no sharing)

Read write sharing: Add another level of indirection to AM entry – need to be able to share protection and see all changes



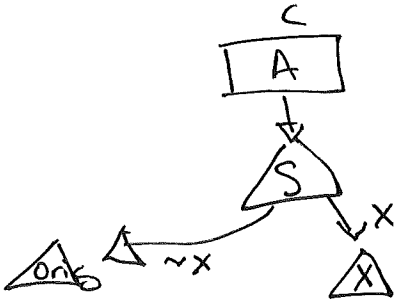


conceptually (can be optimized when full overlap

1) All read same data, nothing changes

- can't get rid of orig. when all overlap

2) Child writes X→x' (Child see x', parent X)



Inefficient:
C forks another child
long chain

3) Parent writes y

What is the purpose of the **pmap?** Why doesn't pmap need to contain every virtual to physical mapping? What is the minimal pmap?

Represents machine dependent parts; manages whatever the hardware interacts with

Only needs to contain what is needed for efficient, correct operation, does need to be complete – just a kcache

some va → pa mappings?

KEY to supporting sparse, large vm while reducing size of page tables

Can get other mappings by going through address map and resident page table map

Minimal pmap: TLB entries

11. How were large page tables dealt with in VAX/VMS? How does Mach on VAX deal with large page tables?

VMS: Place page tables in kernel space so could be paged ~~out~~ themselves

Mach: pmap does not all have to be active; only those pages currently resident are interesting; address map handles sparse spaces better than page tables

Use Mach address map on page fault, fill in pmap

12. To put all the data structures together, describe what happens on a "page fault".

Implies translation is not in pmap

1) Check if in physical memory; how?
    a. Va->memory object->PA
    b. Use address map to find memory object
    c. Use resident page table info – object/offset hash bucket to get physical page
2) If present, inform pmap

3) Not in physical memory
    a. Get free page (from res pt free list)
        i. If none? Look at reference bits for pages and make some free (might require writing out modified page)

    b. Read data ~~from memory object~~ pager

    c. Put allocated page in:

        i. Resdient page table: Memory object list
        ii. Alloc list
        iii. Object/offset hash

    d. Update pmap to PA

13. Conclusion?

Influential microkernel
Most well-known, full developed
Huge research project, commercially available
Influenced others, MacOS X

Two specific contributions:

IPC well-done – copy on write and ports

Portability – good job abstracting independent and dependent components – probably only care about this when really have to deliver OS on many different HW platforms

Maintainability matters

Problem: VM performance isn't critical
Nice functionality, good modularity
No benefit to user-provided pagers