



11/2

- H. M. Levy and P. Lipman, "Virtual Memory Management in VAX/VMS", IEEE Computer, Vol. 15, No. 3, March 1982, pp.35-41.

Q: *The paper states, "VAX/VMS, then, is a collection of procedures that exist in the address space of each process." Explain in your own words what this statement means.*

- Richard Rashid, Avadis Tevanian, Michael Young, David Golub, Robert Baronn, David Black, William Bolosky, and Jonathan Chew, "Machine-Independent Virtual Memory Management for Paged Uniprocessor and Multiprocessor Architectures," In *Proceedings of the Second International Conference on Architectural Support for Programming Languages and Operating Systems*, October 1987, pp. 31-39.

Q: *Why does Mach support copy-on-write, and how does it implement it?*

## Background

- VAX: Virtual Address Extension
- VMS: Virtual Memory System
- Both VAX and VMS were very influential to today's systems
  - it was a powerful minicomputer at the time
    - it was affordable by CS departments at universities
  - Unix was ported to the VAX, hence VAX+UNIX became a popular research platform
- VMS
  - de-facto virtual memory system for Unix as well
  - Cutler was one of the VMS project leaders
    - Bill Gates hired Cutler to lead the design and development of the NT kernel
    - many aspects of the NT kernel have shades of VMS, even today

2

CSE221 - Operating Systems,  
Yuanyuan Zhou

## Main Point

- What is the main challenge the paper describes?
  - mechanisms and policies for VM system for minicomputers to support a wide range of application requirements
  - timesharing, real-time, batch

4

CSE221 - Operating Systems,  
Yuanyuan Zhou

## Address Space

- What are the four regions of a VAX/VMS address space?

P0

...

P1

...

Sys

...

Res

P0: program region (code/data)

P1: program control (stack, command editor)

Sys: system address space -- the OS

Res: reserved (not used)

11/2/21

CSE221 - Operating Systems,  
Yuanyuan Zhou

## Optimizations

- Paper describes three techniques for VM performance:
  - 1) local page replacement
  - 2) page caching (aka free list mechanism)
  - 3) clustering
- What is local replacement, and why is it used?
  - each process gets a max resident size of memory
  - when it reaches max, it replaces from its own set of pages
- Why?
  - isolation from rest of system
  - memory hog cannot use up resources
  - threshold can be changed according to use of machine

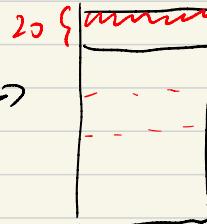
## Benefits of Such Layout

- What is interesting about the addr space layout?
  - OS is essentially just an extension of user address space of every process
  - convenience: OS can directly access user code and data
  - note kernel stacks in user address space (user incurs cost)
- Does it mean we have many COPIES of OS?

Still one copy  
mapped into space

## VM Hardware

- What VM hardware is used?
  - page table
    - page tables do not have holes themselves
    - what are the implications for addr space usage?
      - cannot efficiently support sparse address spaces, e.g., multiple stacks for threads, shared memory, shared libraries, etc.
- TLB



But you may use a little of it.

## User vs. Kernel

- In many respects, user and kernel page tables are similar. But how are they different?
  - user page tables exist in kernel address space
  - kernel page tables exist in physical memory space

kernel cannot be paged out

## Interpreter

- There is something interesting about the command interpreter on VAX/VMS. What is it?
  - command interpreter is a part of each user's address space
  - how is this different from Unix/Windows?

they are in diff address space

## Replacement

- What is their replacement policy?
  - FIFO
- What did you learn in undergrad OS about replacement policies?
  - FIFO not a great performer
- Why did they use it?

It can't guarantee. the hit rate will at least the same even when # of ram ↑

Because the first one may not be the LRU, etc.

easy to implement :).

why did they do ...

- very low overhead algorithm, rely upon page caching (free list) to recover performance w/r/t LRU

CSE221 - Operating Systems

a list of possible removed section

If being access again, being moved out.

## Page Caching

- What is page caching, and how is it implemented?
  - Similar to hardware's victim cache or L1/L2 cache
  - evicted pages are not immediately reused
  - two lists
    - free list: clean pages evicted from a process resident set
    - modified list: dirty pages
  - a fault to one of these pages brings it back into resident set
    - big savings on dirty pages (no writes)
    - also called "second chance" (sound familiar from undergrad OS?)
      - Approximate LRU algorithm
  - performance evaluation found 50% of faults were to pages on lists

## Swapper

- Swapper swaps entire processes and process data structures to and from disk
  - maintains highest priority processes in memory
  - must have enough physical memory for resident set size for swapped process to come back in

→ lower the page fault rate.

## Page Clustering

- What was the motivation for clustering, and what is it?
  - small page sizes: 512 bytes
    - PDP-11 compatibility (ah, the bane of OSes...)
    - supercomputer market at the time, anticipated to be much more successful
    - and it finally has, 30+ years later...
  - inefficient for I/O
  - want to do I/O in large chunks
  - clustering emulates large pages & helps on both reads and writes
  - Reads: demand paging from executable file: read many pages
  - writes
    - write modified pages in one fell swoop (~100 pages at a time!)
    - attempt to write virtually contiguous
    - helps do cluster reads upon subsequent fault
    - theme we see later for LFS

Batch.



Mark.

## Background

- The CS academic research community does not highlight IBM much, but IBM has innovated and had an impact over a long time.
- IBM RT/PC (RISC Technology Personal Computer)
  - IBM's early attempt to capitalize on RISC technology that they had developed very early on with 801 chip (before Berkeley and Stanford projects)
  - Led to IBM's development of the POWER architecture
    - first system was RS6000
    - nice machine, but not a commercial success in mainstream
  - But that led to PowerPC architecture
    - very successful mainframe
    - adopted by Apple (later abandoned for x86)
    - NT initially ran on PowerPC
    - still very successful in embedded systems
    - if you have an Xbox360 or PS3, you own a PPC machine!

11/2/21 CSE221 - Operating Systems

## Mach

- Mach as a research OS has been very influential as well
  - later versions of Unix supported some Mach features
  - Apple's Mac OS X is a blend of the Mach kernel and BSD
    - originally from NeXTStep, which then migrated to Apple when Steve Jobs came back to Apple
    - Avie Tevanian, one of the co-authors, went to Apple after graduating (eventually became CTO)
  - Microsoft NT hardware abstraction layer derived from Mach's example
- Rick Rashid left CMU ~1991 and founded Microsoft Research. MSR is the largest CS research organization in the world (800+ PhDs).

CSE221 - Operating Systems

## Main Idea

*virtual memory*

- Main Idea?
  - maintain all VM state in **machine-independent** module
  - treat hardware page tables/TLBs as caches of machine-independent info
- From a high-level, how is a Mach virtual address space different than a VAX/Unix VAS?
  - can allocate any region in virtual address space (large, sparse VAS)
  - children can inherit regions for sharing (copy-on-write, too)
  - memory mapped files
  - user-level pagers and backing store

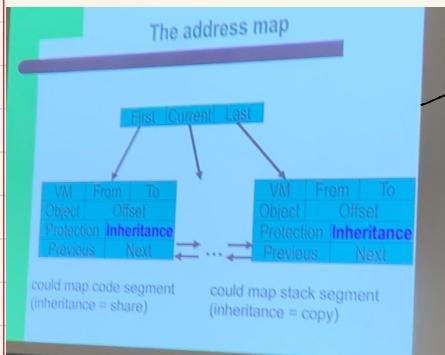
*copy-on-write.*

It is microkernel

## Key Data Structures

- Key data structures in Mach VM
  - address maps
  - memory objects
  - resident page table
  - pmap
- What is the resident page table?
  - keeps track of every resident page in memory
  - linked by
    - memory object using the page
    - allocation queue if not being used
    - hash table for fast page fault handling

11/2/21 CSE221 - Operating Systems, Version 2.0



make use of sparse  
Address Space.  
Linked List.

### The Memory Object

- Mach's memory abstraction.
- Represents a contiguous block of virtual memory (e.g. a file).
- Tasks access by mapping object (or portion of) into their address space.
- Contains any resident pages and port for the backing store.
- Physical memory is not allocated until pages are accessed.
- Each object has a designated memory manager (or pager).

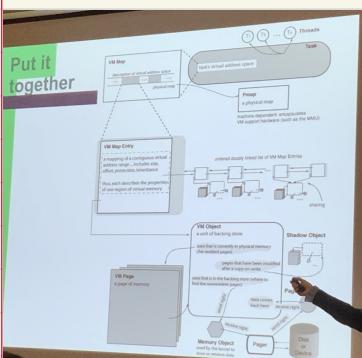
The diagram illustrates the hierarchy of memory objects. It shows the following components and their relationships:

- An "Address map" points to a "Port References" box.
- The "Port References" box points to a "memory Mgr" box.
- A person's hand is pointing at the "memory Mgr" box, with the word "Backing Store" written next to it.

each memory section may have different pager  
user pager will be called to extract when necessary (e.g. encrypted)

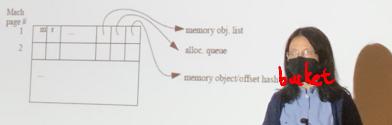
## Address Map

- What is an address map?
  - maps virtual address to memory objects
  - list of address map entries
    - maps ranges of virtual addrs to range of memory objects
  - used to
    - support various VM operations on address ranges
    - page fault lookups
    - copy/protection ops on VA ranges
    - allocation/deallocation of VA ranges



## Resident Page Table

- Used by kernel to maintain state of **physically resident pages** (virtual pages currently in physical memory)
- Page size is somewhat H/W dependent, but scalable (must be power of 2 multiple of machine-dependent size).
- Provides fast lookup of Virtual page on page-fault using **bucket hash** keyed on MemObject::offset
- Page entries also threaded to other lists:



## Inheritance (I)

- After a regular UNIX fork()
  - code segment is shared between parent and child
  - child inherits a copy of data segment of parent
- Mach inheritance attribute specifies if pages in a given range of addresses are to be shared, copied or ignored

T  
one changed  
other see

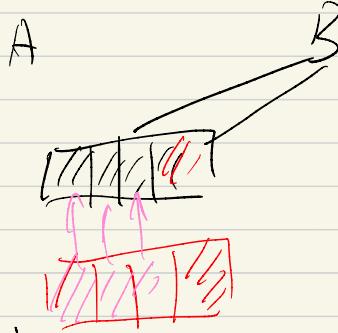
Copy-on-write.

## Inheritance (II)

- Pages of a mapped file are **always shared** between parent and child to preserve file sharing semantics
- Pages in the data segment can either be
  - "copied" to maintain UNIX fork() semantics
  - **shared** if we want to create a thread instead of a regular UNIX process

## Address Map: copy-on-write

- Read only case (lazy-copy)
- Supports memory-sharing across tasks



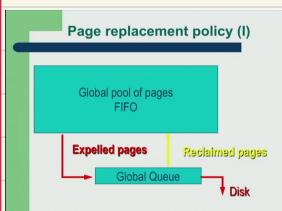
## Address Map: copy-on-write

- Copy after a write, **shadow object** created.
- Shadow object contains only the modified pages, references original object for remainder.

## pmap

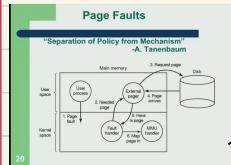
- What is the purpose of the pmap module?
  - module implementing all hardware-dependent VM data and code
  - Each hardware is different
  - very little knowledge of Mach data structures
  - essentially acts as a cache of dependent info
  - Can be rebuilt from Mach data structures

17 11/21 CSE221 - Operating Systems, Yuanxuan Zhou



## Page replacement policy (II)

- Similar to that of VAX VMS
  - Requires little hardware support
- Major change is global FIFO pool replacing resident sets of all programs
  - Much easier to tune
  - Can use external pages



## External Pages

- One associated with each memory object, can be user-defined for specific applications
- Default Mach page handler also provided (at user-level)
- If user sends a pagefault "requested" to a user-level pager, it can decide where to pagefault
- If pager is uncooperative, then default pager will be invited to perform the pagefault
- Update**
  - VM manager based on application needs (e.g. DBMS)
  - Good for maintaining consistency on multiprocessors
  - Minimizes expansion of the bus when over distributed network
- Diversify**
  - Upcalls from kernel (what if no response?)
  - Low cost of context switching (and amortized?)

## Locks and deadlocks

- Mach VM algorithms rely on locks to achieve exclusive access to kernel data structures
  - Price to pay for a parallel kernel
- To prevent deadlocks, all algorithms gain locks using the same linear ordering
  - Well known deadlock prevention technique

request page from User Space

has to be circular

so mark resource number

only lower number can request higher

1 → 2 → 3 → 4 → 1 → *dead circle.*

## Assessing VM Hardware

- What is the main conclusion of VM hardware assessment?
  - no native support
  - usually designed with specific OS in mind
  - when VM features not designed into OS (e.g., sharing), makes it more difficult (but still possible)
    - e.g., inverted page tables

24

11/27/2021

OSRE301 Operating Systems,  
Version 1.0

→  
on  
different  
CPU, not sync