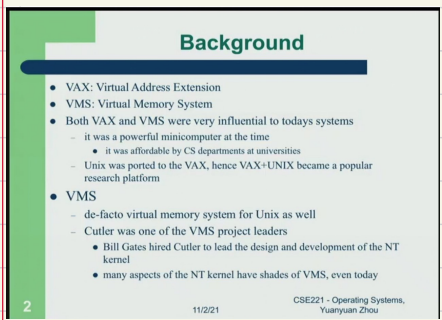
11/2

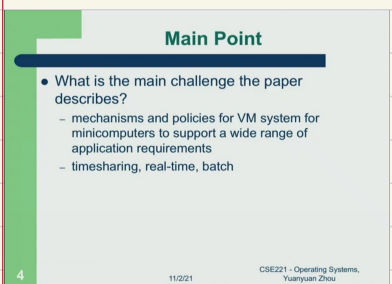
- H. M. Levy and P. Lipman, "Virtual Memory Management in VAX/VMS", IEEE Computer, Vol. 15, No. 3, March 1982, pp.35-41.

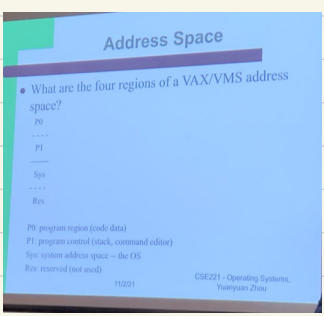
Q: The paper states, "VAX/VMS, then, is a collection of procedures that exist in the address space of each process." Explain in your own words what this statement means.

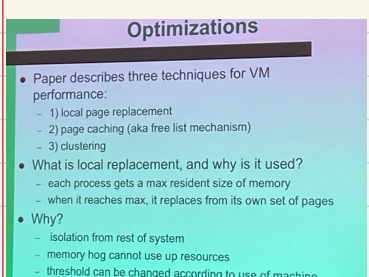
- Richard Rashid, Avadis Tevanian, Michael Young, David Golub, Robert Baronn, David Black, William Bolosky, and Jonathan Chew, "Machine-Independent Virtual Memory Management for Paged Uniprocessor and Multiprocessor Architectures," In *Proceedings of the Second International Conference on Architectural 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 todays 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
11/2/21
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
11/2/21
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

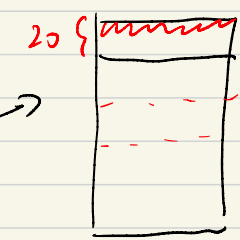
Benefits of Such Layout

- 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 o 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



each section could be 2GB But you may use a little of it.

User vs. Kernel

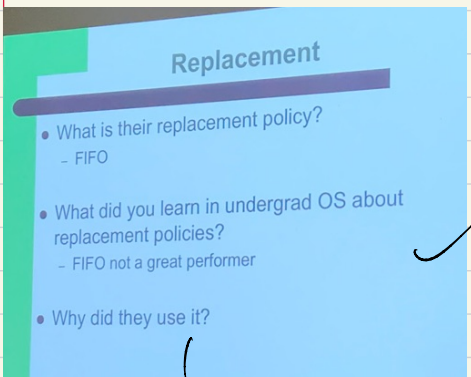
- In many respects, user and kernel page tables are similar. But how are they different?
  - user page tables exist in kernel address space
  - kernel page tables exist in physical memory space

kernel cannot be payed 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?

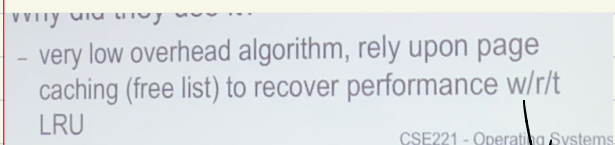
they are in diff address space



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

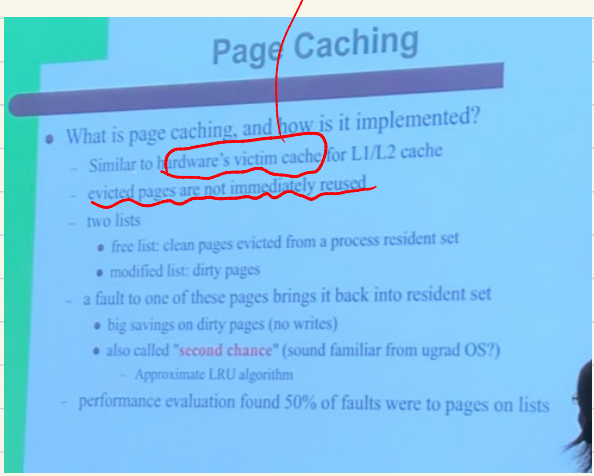
Because the first one may not be the LRU, etc.

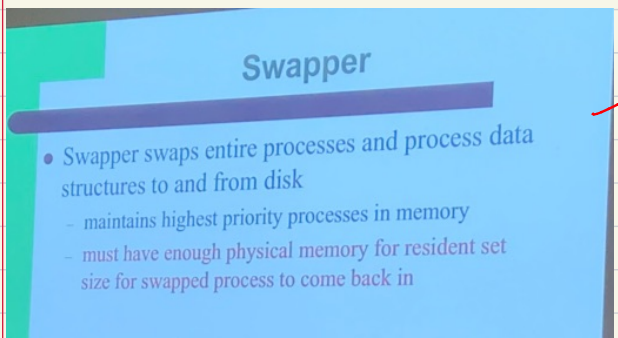
easy to implement :).



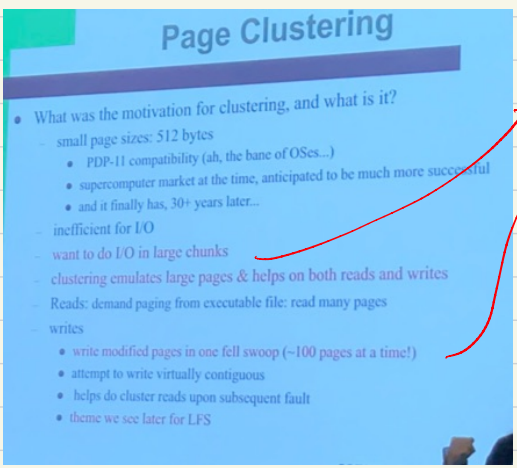
a list of possible removed section

If being access again, being moved out.



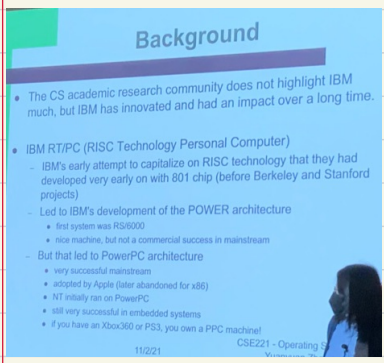


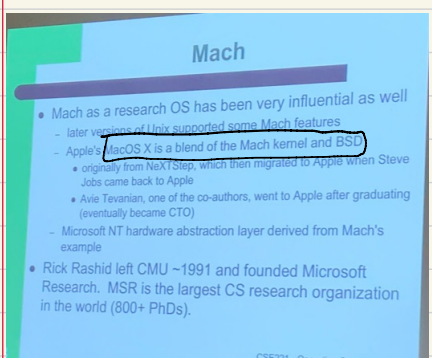
→ lower the page fault rate.

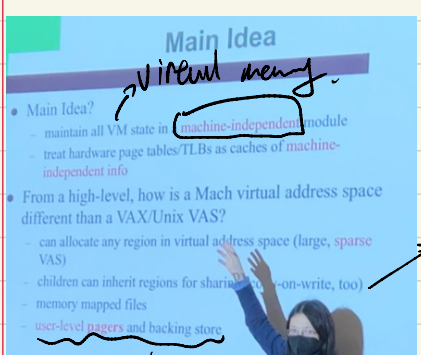


Batch.

Mark.







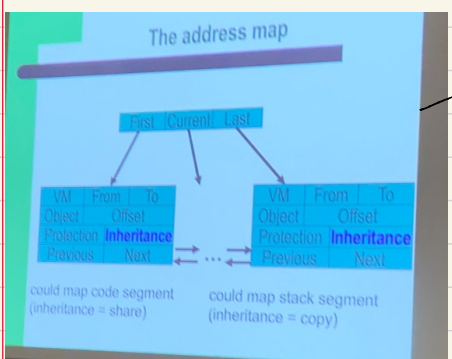
virtual memory.

copy-on-write.

It is microkernel

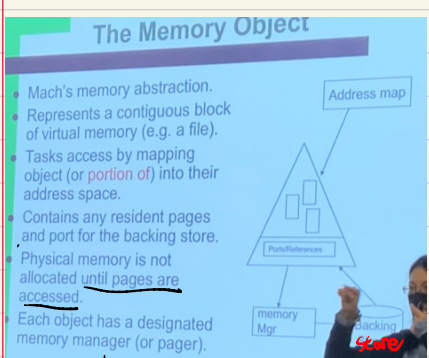
Key Data Structures

- 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



→ make use of sparse Address space.

Linked list.

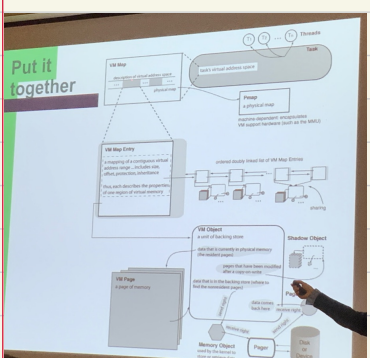


each memory section may have different pager

user pager will be called to extract when necessary (e.g. encrypted)

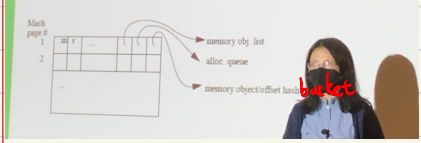
Address Map

- 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**

→ Copy-on-write.

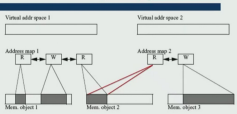
one changed

other see

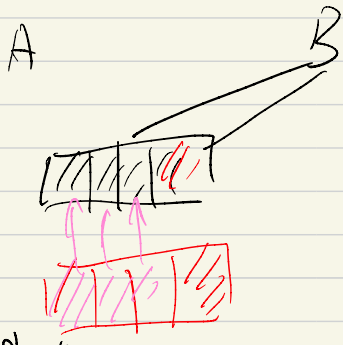
Inheritance (II)

- 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

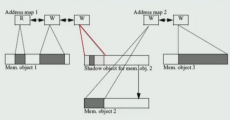
Address Map: copy-on-write



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



Address Map: copy-on-write



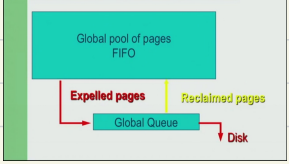
- Copy after a write, **shadow object** created.
- Shadow object contains only the modified pages, references original object for remainder.

shadow object

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

Page replacement policy (I)

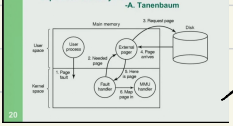


Page replacement policy (II)

- 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 pagers

Page Faults

"Separation of Policy from Mechanism"
-A. Tanenbaum



request pager from User space

External Pagers

- One associated with each memory object, can be user-defined for specific application needs
- Default Mach page handler also provided (at user-level).
- If kernel sends pageout 'request' to a user-level pager, it can decide which page to swap out.
- If pager is uncooperative, then default pager will be invoked to perform the necessary pageout.
- Upside:
  - VM 'tuning' based on application needs (e.g. DBMS).
  - Good for maintaining consistency on multiprocessors.
  - Allowed for expansion of VM sharing over distributed network.
- Downside:
  - Upcalls from kernel (what if no response?)
  - Lots of context switching (well amortized?)

Better customize.

kernel call userspace

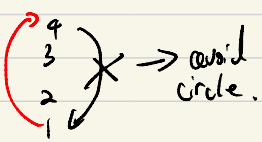
Locks and deadlocks

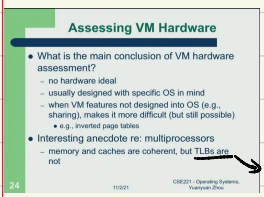
- Mach VM algorithms rely on **locks** to achieve exclusive access to kernel data structures
  - Price to pay for a parallel kernel
- To prevent **deadlocks**, all algorithms gain locks using the same linear ordering
  - Well known deadlock prevention technique

has to be circular

so mark resource number

only lower number can request higher





on different CPU, not sync