Virtual Memory

Robert Grimm
New York University
Altogether Now: The Three Questions

- What is the problem?
- What is new or different?
- What are the contributions and limitations?
Each process has a 32-bit virtual address space
  - Divided into 512-byte pages
Each address consists of
  - 2-bit region identifier
    - P0 program region
    - P1 control region
    - System region
    - Reserved space
  - 21-bit virtual page number
  - 9-bit byte offset within page
Each region has its own page table (why?)
- Base address register
- Length register

Each page table entry consist of
- Valid bit (PTE<31>)
- Protection field (PTE<30:27>)
- Modify bit (PTE<26>)
- OS field (PTE<25:21>)
- Physical frame number (PTE<20:0>)
- System page table is in physical memory
- P0 and P1 page tables are in virtual memory
  - Two accesses per address translation (which?)
- Translation buffer
  - Divided into system and per-process translations
Memory Management Concerns

- Containing the effects of heavily paging programs
- Reducing the cost of program startup
- Reducing the disk workload of paging
- Reducing the processor time searching page tables

- Amplified by minicomputer hardware
  - Slow CPU
  - Slow and limited number of disks
  - But memory size easily changed
The VAX/VMS Pager

- Process-local replacement
  - To contain paging behavior
  - Backed by resident-set list
    - Organized as FIFO, not as LRU
      - How is LRU commonly implemented?
      - Why not LRU?
      - Can any page be evicted?

- Complemented by free and modified page lists
  - Caches of recently used pages, added to tails
  - Simulation studies show that private free lists can approximate LRU replacement with arbitrary precision
  - Are these lists per-process or global?
Clustering of Pages

- To reduce I/O workload, pages should be read/written in groups
  - For reading executables
    - Linker lays out pages consecutively (if possible)
    - Pager reads pages in one operation (if possible)
      - Same for paging file
  - For reading and writing dirty pages
    - Modified page list has low and high watermark
    - Pager writes pages when list reaches high watermark, combining consecutive pages into single writes
      - Virtually or physically consecutive?
Modified Page List + Lazy Writing Is a Good Thing™

- Serves as a cache of recently removed pages
  - Minimal cost for cache hits
- Supports the batching of writes
  - Writes are cheaper
- Supports clustering in paging file
  - Reads are cheaper as well
- Avoid unnecessary writes
  - Page may be used again, program terminates
The Swapper

- Moves entire resident sets between memory and storage
  - Keeps higher priority processes resident
  - Reduces high paging rates in most paged systems
    - Entire resident set is loaded
  - Needs to be careful about ongoing I/O
    - What is the concern?
Back to Memory Management Concerns

- Containing the effects of heavily paging programs
  - How is this achieved?

- Reducing the cost of program startup
  - How is this achieved?

- Reducing the disk workload of paging
  - How is this achieved?

- Reducing the processor time searching page tables
  - How is this achieved?

- Are these techniques effective?
Think Different: Mach’s VM
The Mach Microkernel

- Five core abstractions
  - Tasks, threads, ports, messages, memory objects
- One big challenge
  - Making the system perform well
  - Claim: Integration of VM with messaging is key
Mach Virtual Memory Features and Operations

- **Features**
  - Large, sparse virtual address spaces
  - Copy-on-write
  - Read-write memory sharing
  - Memory mapped files
  - User-provided backing store objects and pagers

- **Operations**
  - Allocating and de-allocating virtual memory
  - Setting protection status of VM
  - Setting inheritance for VM (shared, copy, none)
  - Managing memory objects
Implementation Strategy

- Minimize state in machine-dependent data structures
  - The *pmap*: Hardware-defined physical address map

- Machine-independent data structures
  - The *resident page table*: attributes of physical memory
    - Embedded memory object lists
    - Embedded allocation queues for free, reclaimable, allocated pages
    - Addresses expressed as bytes (why?)
  - The *address map*: doubly-linked list of virtual-to-memory-object mappings
    - Sorted by virtual address
    - Includes inheritance and protection attributes
Machine-independent data structures (cont.)

- *Memory objects*: repositories of actual data
  - Reference-counted
  - Cached (for frequently used memory objects)
  - Controlled by *pager* (e.g., to implement memory-mapped files)

Special support for memory sharing

- *Shadow objects*: list of modified pages for copy-on-write
  - Add quite some complexity to implementation
- *Sharing maps*: level of indirection for read/write shared memory
The pmap Module

- Necessary to interact with real hardware
- But maintains only soft state (!)
  - All page tables can be reconstructed from other VM data structures
- With the exception of kernel mappings
  - Must be accurate
Some Uniprocessor Issues

- VAX has very small pages → very large tables
  - Construct only those that are actually used
- IBM RT PC has inverted page table
  - Need to treat page table as software TLB due to VM aliasing
- SUN 3 has holes in physical memory
  - Need to treat page table as sparse data structure
One Multiprocessor Issue

- How to ensure consistency of mappings across processors?
  - Machines have no support for TLB consistency
  - Mach needs to track TLB contents and propagate changes
    - Forcibly interrupt CPUs on time critical changes
    - Change mappings on timer interrupt for, say, pageouts
    - Allow inconsistency for protection changes
What Do You Think?