Virtual Memory

Robert Grimm
New York University

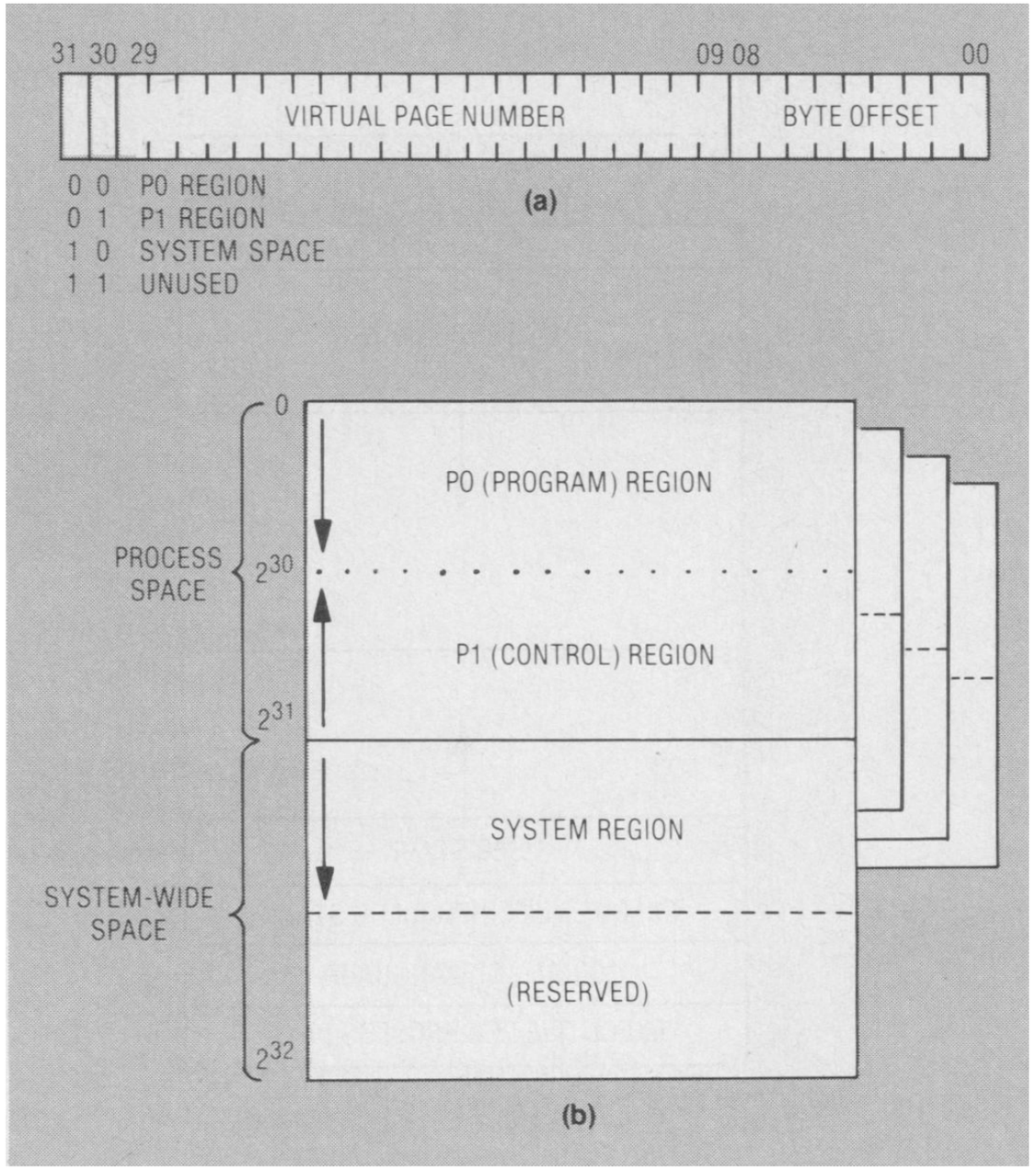
The Three Questions

- What is the problem?
- What is new or different?
- What are the contributions and limitations?

VAX/VMS

VAX-11 Memory Hardware

- 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
    - Reserve space
  - 21-bit virtual page number
  - 9-bit byte offset within page



More on VAX-11 Memory

- Each region has its own page table — why?
  - Base address register
  - Length register
- Each page table entry consists 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>)

Even More on VAX-11 Memory

- System page table is in physical memory
- P0 and P1 page tables are in virtual memory
  - Two accesses per address translation — to what data?
- Translation buffer
  - Divided into system and per-process translations
    - What entries need to be flushed on a context switch?

VM 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
- Supported by process-local resident-set list
  - Organized as FIFO, not as LRU
    - How is LRU commonly implemented?
    - Why not LRU?
    - Which page is evicted on a page fault?
- Complemented by global free- and modified-page lists
  - Caches of recently used pages, which are added to tails
  - Simulation studies show that *private* free lists can approximate LRU with arbitrary precision

Clustering of Pages

- Pages read/written in bunches to reduce I/O workload
  - For reading executables
    - Linker lays out pages consecutively (if possible)
    - Pager reads pages in one operation (if possible)
  - For reading and writing dirty pages
    - Modified page list has low and high watermarks
    - Pager writes pages when list reaches high watermark, combining consecutive pages into single writes
      - Virtually or physically consecutive?

The Benefits of Delay

- Modified page list combined with lazy writing
  - Serves as a cache of recently removed pages
    - Minimal cost for cache hits
  - Supports batching of writes
    - Writes are cheaper
  - Supports clustering in paging file
    - Reads are cheaper as well
  - Avoid unnecessary writes
    - Page may be used again or program terminates

The Swapper

- Moves entire resident sets between memory and disk
  - Keeps higher priority processes resident
  - Reduces high initial paging rates in most paged systems
    - Entire resident set is loaded
  - Needs to be careful about on-going I/O
    - What is the concern?

Back to VM 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?

Think Different: Mach's Virtual Memory

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
    - What is the claim in more detail?
- Oh, and provide all this in a machine-independent fashion...

Features and Operations

- Features
  - Large, sparse virtual address spaces
  - Copy-on-write
  - Read-write memory sharing
  - Memory mapped files
  - User-provided pagers and backing store
- Operations
  - Allocating and de-allocating virtual memory
  - Setting protection status of VM
  - Setting inheritance for VM (shared, copy, none)
  - Managing memory objects (i.e., backing storage)

Implementation Strategy

- Minimize state in machine-dependent data structures
  - *pmap*: hardware-defined virtual-to-physical address map
- Machine-independent data structures
  - *Resident page table*: attributes of physical memory
    - Memory object lists to identify backing store
    - Allocation queues for free, reclaimable, allocated pages
    - Hash table from objects/offsets to pages
  - *Address map*: attributes of virtual memory
    - Doubly linked list of virtual-to-memory-object mappings
      - Sorted by virtual address, includes inheritance/protection atts

More Implementation

- Machine-independent data structures (cont.)
  - *Memory objects*: repositories of actual data
    - Reference counted
    - Cached after last unmapping (for frequently used m.-o.'s)
    - 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
  - *Sharing maps*: level of indirection for read/write shared memory
    - Taken together, they add quite some complexity

The pmap Module

- Necessary to interact with real hardware
- But maintains only *soft state*
  - All page table info 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 page tables
  - Construct only those entries that are actually used
- IBM RT PC has inverted page table
  - Mapping physical-to-virtual addresses
  - Need to treat page table as software TLB — why?
- 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 nodes?
  - 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?