Nathan Tung

CS111 Chapter 5.4, 6.2 Outline #13

Chapter 5, Section 5.4 (pages 242-255). Chapter 6, section 6.2.2-6.2.9 (pages 323-347).

- Virtualizing Memory
  - Memory and address space are limited
    - For a program to grow domain, new memory may have to be allocated
  - Most computer systems virtualize memory with two features:
    - Virtual addresses: if using virtual addresses, memory manager can grow/move domains in memory behind the program's back
    - Virtual address spaces: multiple programs might need more than an address space at once; a single program can obtain own (virtual) address space/enable thread to start at certain address
  - Virtual memory manager: memory manager that virtualizes memory
    - Incorporates main features of domains: controlled sharing and permissions
- Virtualizing Addresses
  - Threads issue virtual addresses when reading/writing to memory
  - Memory manager translates this to physical address (bus address of location in memory or register on device controller)
  - Translating addresses provides flexibility
    - Virtual addresses can map to same physical address with different permissions
    - Physically addresses can have different widths
    - Virtual addresses can postpone physical memory allocation until referenced
  - Design principle: decouple modules with indirection
    - Virtual memory manager is a layer of indirection between processor and memory system
    - It translates between virtual and physical addresses, allowing rearranged data in memory
  - Name space of virtual address is created on top of that of physical address, essentially translating it
  - With address translation of virtual addresses, the application doesn't need to know that the memory manager moved/copied contents of its domain in order to grow it
  - Unfortunately, virtual addresses comes with a cost to complexity and performance
- Translating Addresses Using a Page Map
  - Naïve way: use a map to link virtual and corresponding physical addresses; requires lots of memory
  - Better way: use a page map, or an array of page map entries
    - Entries translate a fixed-sized rang e of contiguous bytes of virtual addresses (page) to the corresponding range of physical addresses (block)
    - Threads see memory as a set of contiguous pages
    - Virtual address is a name overloaded with structure of two parts: page number and byte offset
      - Page number selects an entry in page map (a page); byte offset selects byte in that page
    - Block acts like a container of a page; physical memory is a contiguous set of blocks with pages
      - Pages don't have to be contiguous
    - Simplifies memory management: manager can allocate block anywhere and insert mapping
    - Physical address also consists of two parts: block number and byte offset within block
    - Translation from virtual to physical address now takes two steps
      - Virtual memory manager translates virtual address page number to block number
      - It then computes physical address by concatenating block number with byte offset
  - Implementations of page map
    - Page table (array implementation): used when pages have associated block
      - Translates address by using page number as index into page table to find block number
      - Manager computes physical address by concatenating byte offset with block number
  - Page-map address register: reserved processor register that holds physical address of base page map
    - Write/modify in kernel mode only
  - When thread wants new/more domain, kernel can allocate unused block and insert into page map
    - High level of indirection; the running threads don't know what the kernel did
- Virtual Address Spaces
  - Many processors have virtual address space that is too small to hold all active modules and their data
    - Solution is to virtualize the physical address space

- Primitives for Virtual Address Spaces
  - Virtual address space tricks application into thinking it has complete address space to itself
  - Virtual memory manager can implement virtual address space using page map and interface:
    - CREATE_ADDRESS_SPACE, ALLOCATE_BLOCK, MAP, UNMAP, FREE_BLOCK, DELETE_ADDRESS_SPACE
    - Thread can allocate its own address space or share with other threads
    - Process: combination of single virtual address space shared among threads
      - "Process" has been overused and refers to so many things; one thread in private address space, group of threads in private address space, or shared thread
      - Use thread and address space instead (UNIX)
  - Physical block can be shared in reference when It is found in two page maps
  - Domain permissions placed in page-map entries allow read/write permissions
  - Linear page table is like array indexed by page number and stores corresponding block number
- Kernel and Address Spaces
  - Option 1: address space includes mapping of kernel into address space
    - Easy switching between user program and kernel (only user-mode bit changes)
    - Kernel page permissions are set to KERNEL-ONLY for protection
    - Kernel easily reads data from user program structures due to shared address space
    - Disadvantage: reduces available address space for user programs
  - Option 2: memory manager gives kernel separate address space (from user-level threads)
    - Extend svc instruction to switch page map address register to kernel's page map during kernel mode; during user mode, change back page-map address register
    - Memory manager must be able to create new address spaces for user programs, etc.
    - Include page tables of user address spaces in kernel address space
    - Page table is smaller than address space it defines; option 2 wastes less address space
    - Separated address spaces means user programs must pass arguments to supervisor calls by value or kernel must use an involved method for copying data over
- Discussion
  - Advantages of virtual
    - Programs don't need to be compiled independently in position (unimportant though)
  - Disadvantage
    - Sharing can be confusing and inflexible (ex: shared block, mapped by 2 threads in 2 address spaces at different virtual addresses)
    - Different address space threads share can only share objects mapped at page boundary
  - Position-Independent Programs
    - Programs that can be loaded at any memory address; compiler uses relative addresses
- Hardware vs Software and Translation Look-Aside Buffer
  - No real answer; tradeoffs between hardware and software
  - Hardware reduces opportunities for OS to exploit translation between virtual/physical addresses
  - Cache entries of page map to avoid additional bus references
  - Use translation look-aside buffer (TLB) as cache memory of translations
    - When translation not found in TLB, processor gives TLB miss exception
- Segments
  - Segments provide each object with virtual address space from 0 to size of object
  - Allows threads to share memory at object granularity rather than blocks (more flexible)
  - Segment descriptor table holds segment descriptors corresponding to processors
  - Advantage: designer doesn't need to predict max size of objects that grow dynamically

- Multilevel Memory Management using Virtual Memory
  - Multilevel memory: system of devices from more than one level
  - Fast, small, expensive vs. big, slow, cheap
  - CPU registers → Level 1 cache → Level 2 cache → Main memory → Magnetic disk → Remote storage
  - Two ways to management multilevel memory
    - Let programmer decide which memory to use for data
    - Use automatic management (subsystem independent of application program)

        - Good performance on average
        - Relieves programmer of need to write specifics
    - One-level store: combine automatically managed multilevel memory system with virtual memory manager so READ/WRITE applies to entire memory system
    - Decouple modules with indirection: achieve this using multilevel memory manager to do the above
    - Indirection exception: memory reference exception indicating memory manager cannot translate a certain virtual address
    - Other options
        - Memory-mapped files: application can read/write to opened/mapped file as if it were RAM
        - Copy-on-write: copies a page that is changed
        - On-demand zero-filled pages: saves RAM/storage space by filling in page with zeros
        - One zero-filled page: implementation of the above two
        - Virtual shared memory: virtual memory manager fetches non-local page over network
    - Adding Multilevel Memory Management to Virtual Memory
        - At any instant, only some pages in page map are in RAM due to limited capacity
            - Resident bit tells us if it's in RAM (1 = true)
        - Missing-page exception (or page fault) can interrupt processor via virtual memory manager
        - When blocks in RAM are occupied with pages, multilevel memory manager selects some page from RAM and removes it
            - Removed page is the victim; algorithm is the page-removal policy
        - Multilevel memory manager might do implicit I/Os in addition to the program's explicit ones
    - Separate Mechanism from Policy
        - Separating missing-page mechanism from page replacement policy
    - Analyzing Multilevel Memory Systems
        - Fast level devices in a two-level memory system are primary devices (vs. secondary)
            - Primary is usually RAM, secondary is slower RAM or magnetic disk
        - Cache and virtual memory are very similar
            - Only difference: cache uses namespace of secondary memory device to identify memory cells, while virtual memory uses primary memory device
- Locality of Reference and Working Sets
    - Two level memory
        - AverageLatency = R_bit * Latency_primary + R_miss * Latency_secondary
        - If primary/secondary had equal frequency in access, average latency would be proportional to the number of cells of each device
    - Locality of reference: concentration of access into small but shifting locality
        - Think of running application as generating virtual address stream, or reference string
        - Temporal locality: reference string has closely spaced references to same address
        - Spatial locality: reference string has closely spaced references to adjacent addresses
        - Example: programs are sequences of instructions, usually stored adjacent to each other
        - Example: data structures have adjacent references
    - Working set: set of references of application in a given interval
    - Thrashing: repeated back and forth movement of data between primary/secondary devices
        - Due to working set being much larger than primary device; try to minimize this
- Multilevel Memory Management Policies
    - For items characterize each level of multilevel memory system
        - String of references: sequence of page numbers in virtual memory system
        - Bring-in policy: on-demand; when a page is used, bring to primary device if not there
            - Isn't needed for secondary device in two-level memory system
        - Removal policy: choose a page (victim) to evict to make room for new page
            - Isn't needed for secondary device in two-level memory system
        - Capacity: number of primary (or secondary, for secondary capacity) memory blocks
    - FIFO page-removal policy
    - Belady's anomaly: unexpected increase of missing-page exception numbers with larger primary device capacity

- Goal of multilevel memory management policy is to select pages to remove that minimize number of missing-page exceptions in future
  - Optimal (OPT) page-removal policy is unrealizable
- Least-recently-used (LRU) page-removal policy is popular
- Demand algorithms: algorithms that require new page to be the only page that moves in and that only one page move out
- Most-recently-used (MRU) page-removal policy gives fewer exceptions than LRU sometimes
- In practice, LRU is robust and does better than MRU usually
- OPT > LRU > MRU > FIFO

- Comparative Analysis of Different Policies
  - Two things that affect performance
    - How large primary memory device should be
    - Which page-removal policy to use
  - Chosen based on analyst's trace of reference string on a program
  - Subset property
  - Stack algorithms: page-removal policies that maintain subset property
  - OPT is stack algorithm and optimal
    - If OPT removes a page from primary memory, it always chooses the page not needed for the longest time in the future
      - Total ordering of pages is independent of primary device capacity, so stack algorithm
      - Thus, for a reference string, set of pages in primary device of capacity m is always subset of the set of pages in primary device of capacity m+1 (subset property)
- Other Page-Removal Algorithms
  - Clock page-removal algorithm depends on the referenced bit
    - Removes page in first block that has FALSE referenced bit (if available)
    - Provides rough approximation to LRU; divides pages into two categories
      - Those used or not used since last sweep
  - Random removal policy for the translation look-aside buffer is simpler
    - Minimal state to implement; penalty for wrong translation is small
  - Direct mapping is a stateless policy
    - Choose page as the one located in block n modulo m (n is secondary device block number, m is number of blocks in primary device)
- Other Aspects of Multilevel Memory Management
  - Bring-in policy: pages are moved to primary device only when application tries to use them
    - Demand paging systems
  - Prepaging: predict which pages might be needed, bring them into primary device
  - Both speculate; demand paging assumes other bytes on the page will be used; prepaging assumes the application will used the prepaged pages
  - Working set might not all fit in primary set; swapping is needed to avoid thrashing
    - Multilevel memory manager moves pages out of primary device in batch
    - Batch of writes to disk are faster than a series of single-block writes