Lecture 19: Main Memory/Virtual Memory

• Last Time
  - Main memory

• Today
  - DRAM paper discussion
  - Virtual memory
  - Page tables and TLBs

Simple View of Memory

- Single program runs at a time
- Code and static data are at fixed locations
  - code starts at fixed location, e.g., 0x100
  - subroutines may be at fixed locations (absolute jumps)
  - data locations may be wired into code
- Stack accesses relative to stack pointer.
Running Two Programs (Relocation)
No Protection

- Need to relocate logical addresses to physical locations
- Stack is already relocatable
  - all accesses relative to SP
- Code can be made relocatable
  - allow only relative jumps
  - all accesses relative to PC
- Data segment
  - can calculate all addresses relative to a DP
    - expensive
  - faster with hardware support
    - base register

Base-Register Addressing

- Add a single base register, BR, to hardware
- Base register loaded with data pointer (DP) for current program
- All data addresses added to base before accessing memory
  - Can relocate code too
- Often implemented with a three-input adder
- Need to bypass base register to access system tables for program switching
  - a place to stand
Base Register Addressing

System code handles switching between programs
System table contains
Base address of each program
Saved state of non-running programs

Providing Protection Between Programs (Length Registers)

- Add a Length Register LR to the hardware
- A program is only allowed to access memory from BR to BR+Length-1
- A program cannot set BR or LR — they are privileged registers
- But how do we switch programs?
This is Segmentation!

- **Operating system creates segments**
  - Gives them to users

- **Segments**
  - Access attributes (read, r/w, execute)
  - Size (and a way to test for illegal access)
  - Relocation (but not in a convenient way)

- **Is there a better way that:**
  - Simplifies protection
  - Enables relocation
  - Extends the physical memory capacity
A Load to Virtual Memory

- Translate from virtual space to physical space
  - $VA \Rightarrow PA$
  - May need to go to disk
- Different virtual address spaces for different processes

LW R1, 0(R2)

CPU \rightarrow \text{Cache} \rightarrow \text{Physical Addr.} \rightarrow \text{Translate} \rightarrow \text{Virtual Addr.}

Virtual Addr. 32 bits

Physical Addr. 26 bits

22 bits

Process 1

DRAM 64MB

Process 2

Paging: Main Memory as a Cache for Disk

- 32 bit addresses = 4GB, Main Memory = 64MB
- Dynamically adjust what data stays in main memory
  - Page similar to cache block
- Note: file system $\gg$ 4GB, managed by O/S
Virtual Addresses Span Memory+Disk

- Mappings changed dynamically by O/S
  - In response to users data accesses
  - OS triggered by hardware

Virtual Address Translation

- Main Memory = 64MB
- Page Size = 4KB
- VPN = 20 bits
- PPN = 14 bits

Translation table
  - aka “Page Table”
Summary

- Virtual memory provides
  - Illusion of private memory system for each process
  - Protection
  - Relocation in memory system
  - Demand paging
- But - page tables can be large

Page Table Construction

- Page table size
  - $(14 + 1) \times 2^{20} = 4\text{MB}$
- Where to put the page table?
Paging and Protection

- How to ensure that processes can’t access each other’s data
  - Put them in separate virtual address spaces
  - Control the mappings of VA to PA for each process
    - Separate page tables
- How can you share data between processes
  - Give them each a VA mapping to the same PA
    - Similar entry in each process’ page table

What if Data is Not in DRAM?

1) Examine page table
2) Discover that no mapping exists
3) Select page to evict, store back to disk
4) Bring in new page from disk
5) Update page table
VM Requires

- **Restartable (or resumable) instructions**
  - must be able to resume program after recovering from a page fault
- **Ability to mark a page *not present***
  - and raise a page fault when referencing such a page
- **(Optional) Maintain status bits per page**
  - R - referenced - for use by replacement algorithm
  - M - modified - to determine when page is *dirty*
Page Frame Management

- OS maintains
  - page table for each user process
  - page frame table
  - free page list
    - pages evicted when number of free pages falls below a low water mark.
    - pages evicted using a replacement policy
      - random, FIFO, LRU, clock
    - if M-bit is clear, need not copy the page back to disk

Page Management and Thrashing

- Need to keep a process’ working set in memory or thrashing will occur
- Find working set size by increasing page frame allocation until PF/s falls below limit
- Swap out whole process if insufficient page frames for working set

Reference four pages in sequence, mapped to three page frames
Page Table Organization

- Flat page table has size proportional to size of virtual address space
  - can be very large for a machine with 64-bit addresses and several processes
- Three solutions
  - page the page table (fixed mapping)
    - what really needs to be locked down?
  - multi-level page table (lower levels paged - Tree)
  - inverted page table (hash table)

Multi-Level Page Table

e.g., 42-bit VA with 12-bit offset
10-bits for each of three fields
1024 4-byte entries in each table (one page)
Inverted Page Tables

- Store only PTEs for pages in physical memory
- Miss in page table implies page is on disk
- Need KP entries for P page frames (usually K > 2)

How Long does it Take to Access VM?

Best Case

Worst Case

- Problems
  - Multiple memory and potentially disk accesses
  - Where does the cache fit in?
- Let’s accelerate this process!
Translation Lookaside Buffers

- Store most frequently used translations in small, fast memory
- Valid, Writeable, Referenced, Modified
  - Access protection
  - Replacement strategies

Page = VPN
Frame = PPN

“Rare” Behavior in VM system

- TLB Miss
  - Translation is not in TLB - but everything could be in memory
  - Two approaches
    - Hardware state machine walks the page table
      - fast but inflexible
    - Exception raised and software walks the page table
- Page Fault
  - Entry not in TLB and target page not in main memory
- Worst case
  - Page fault and page table and target page
Reducing TLB misses

- **Same type of optimizations as for cache**
  - Associativity (many TLBs are fully associative)
  - Capacity - TLBs tend to be 32-128 entries
- **Adjust page size**
  - Small pages
    - Reduces internal fragmentation
    - Speeds page movement to/from disk
  - Large pages
    - Can cover more physical memory with same number of TLB entries
  - Solution - typically have a variable page size
    - Select by OS, 4KB-256KB (superpages)

Virtual Memory + Caching

- **Conflicting demands:**
  - Convenience of flexible memory management (translation)
  - Performance of memory hierarchy (caching)
- **Requires cooperation of O/S**
  - Data in cache implies that data is in main memory
- **Combine VM and Caching**
  - Where do we put the Cache and the TLB????
Physically Addressed Cache

- Translate first from VA $\rightarrow$ PA
- Access cache with PA
- Problems?

Virtually Addressed Cache

- Access cache first
- Only translate if going to main memory
- Problems?
Aliasing

- Can occur when switching among multiple address spaces
- Synonym aliasing
  - Different VAs point to the same PA
  - Occurs when data shared among multiple address spaces
  - One solution - always translate before going to the cache
- Homonym aliasing
  - Same VA point to different PAs
  - Occurs on context switching
  - Two solutions:
    - Flush TLB on process switch/system call
    - TLB includes process ID

Best of Both Worlds:

Virtually addressed, Physically Tagged
- Parallel Access
- Eliminate synonym problem
Virtual Index, Physical Tag

Other Aliasing Solutions

- Note virtually indexed/physically tagged put constraints on cache capacity, page size, etc.
- Other solutions:
  - 21264: 8KB pages, 64KB i-cache, 2-way set associative
    - Aliases could reside in 8 different places in cache
    - On cache miss, invalidate any possible aliases in cache
  - Intel Pentium 4
    - Virtually indexed/virtually tagged cache
    - Check for TLB misses off line (roll back if necessary)
Virtual Memory Summary

• Relocation, Protection
• Access memory >> DRAM capacity

• Translation
  - From large VA space to smaller PA space
  - Page tables hold translations

• Provides
  - Separation of memory management from user programs
  - Ability to use DRAM as cache for disk
  - Fast translation using Translation Lookaside Buffer (TLB)
    - Cache for page table
  - Speed - translate in parallel with cache lookup

Summary

• Page table management
  - Flexibility in the system
  - Need to avoid nasty issues such as aliasing

• TLBs
  - Cache for page table
  - Need to keep fast so as to not increase AMAT

• Next time
  - Memory system case studies
  - Virtual machines overview
  - Reading: Appendix C.4 - C.6