Virtual Memory

Reading

- Sections 5.4, 5.5, 5.6, 5.8, 5.10
The Memory Hierarchy

Virtual to Physical Mapping

- Exploit program locality at page granularity
- Program can be larger than memory
- At any point in time, the program is in memory+disk
Virtual Memory

- Use main memory as a “cache” for secondary (disk) storage
  - Managed jointly by CPU hardware and the operating system (OS)

- Programs share main memory
  - Each gets a private virtual address space holding its frequently used code and data
  - Protected from other programs

- CPU and OS translate virtual addresses to physical addresses
  - VM “block” is called a page
  - VM translation “miss” is called a page fault

Address Translation

- Fixed-size pages (e.g., 4K)

- Examples of translation
Address Translation: Concepts

- Offsets within the virtual page and corresponding physical page are the same
- We only need to translate the virtual page number (VPN) to the corresponding physical page number (PPN) also called page frame → effectively a base address

Page Tables

- Stores placement information
  - Array of page table entries, indexed by virtual page number
  - Page table register in CPU points to page table in physical memory
- If page is present in memory
  - Page table entry (PTE) stores the physical page number
  - Plus other status bits (referenced, dirty, ...)
- If page is not present
  - PTE can refer to location in swap space on disk
Translation Using a Page Table

Page table register

Virtual address

<table>
<thead>
<tr>
<th>Virtual page number</th>
<th>Page offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>31 30 29 28 27</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Valid</td>
<td>Physical page number</td>
</tr>
<tr>
<td></td>
<td>12</td>
</tr>
<tr>
<td>Page table</td>
<td></td>
</tr>
</tbody>
</table>

If page is not present in memory

<table>
<thead>
<tr>
<th>Physical page number</th>
<th>Page offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>29 28 27</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Physical address

(9)

Page Fault Penalty

- On page fault, the page must be fetched from disk
  - Takes millions of clock cycles
  - Handled by OS code

- Try to minimize page fault rate
  - Fully associative placement
  - Smart replacement algorithms

(10)
Mapping Pages to Storage

- To reduce page fault rate, prefer least-recently used (LRU) replacement
  - Reference bit (aka use bit) in PTE set to 1 on access to page
  - Periodically cleared to 0 by OS
  - A page with reference bit = 0 has not been used recently

- Disk writes take millions of cycles
  - Write through is impractical
  - Use write-back
  - Dirty bit in PTE set when page is written
Caching PTEs: The Translation Lookaside Buffer (TLB)

- Keep a cache of most recently used PTEs
  - Each PTE corresponds to a “relatively” large part of memory
    - For example, a 16Kbyte page may have 4K instructions
  - A small set of PTEs can cover a large code segment
    - For example, 8 PTEs and 16 Kbyte pages corresponds to a program size of 32K instructions
- The TLB access time is comparable or better than cache access time
- Typically operates as a fully associative cache, but can be implemented as a set associative cache

![A four entry TLB](image)

---

Fast Translation Using a TLB

![Diagram showing translation using a TLB](image)
TLB Operation

- TLB size typically a function of the target domain
  - High end machines will have fully associative large TLBs
- PTE entries are replaced on a demand driven basis
- The TLB is in the critical path

TLB Misses

- If page is in memory
  - Load the PTE from memory and retry
  - Could be handled in hardware
    - Can get complex for more complicated page table structures
  - Or in software
    - Raise a special exception, with optimized handler
- If page is not in memory (page fault)
  - OS handles fetching the page and updating the page table
  - Then restart the faulting instruction
TLB Miss Handler

- TLB miss indicates one of
  - Page present, but PTE not in TLB
  - Page not present
- Must recognize TLB miss before destination register overwritten
  - Raise exception
- Handler copies PTE from memory to TLB
  - Then restarts instruction
  - If page not present, page fault will occur

Page Fault Handler

- Use faulting virtual address to find PTE
- Locate page on disk
- Choose page to replace
  - If dirty, write to disk first
- What about copies in the cache?
- Read page into memory and update page table
- Interaction with the operating system: make process runnable again
  - Restart from faulting instruction
TLB and Cache Interaction

- If cache tag uses physical address
  - Need to translate before cache lookup

- Alternative: use virtual address tag
  - Complications due to aliasing
    - Different virtual addresses for shared physical address

- Example problems

---

2-Level TLB Organization

<table>
<thead>
<tr>
<th></th>
<th>Intel Nehalem</th>
<th>AMD Opteron X4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual addr</td>
<td>48 bits</td>
<td>48 bits</td>
</tr>
<tr>
<td>Physical addr</td>
<td>44 bits</td>
<td>48 bits</td>
</tr>
<tr>
<td>Page size</td>
<td>4KB, 2/4MB</td>
<td>4KB, 2/4MB</td>
</tr>
<tr>
<td>L1 TLB (per core)</td>
<td>L1 I-TLB: 128 entries for small pages, 7 per thread (2×) for large pages</td>
<td>L1 I-TLB: 48 entries</td>
</tr>
<tr>
<td></td>
<td>L1 D-TLB: 64 entries for small pages, 32 for large pages</td>
<td>L1 D-TLB: 48 entries</td>
</tr>
<tr>
<td></td>
<td>Both 4-way, LRU replacement</td>
<td>Both fully associative, LRU replacement</td>
</tr>
<tr>
<td>L2 TLB (per core)</td>
<td>Single L2 TLB: 512 entries</td>
<td>L2 I-TLB: 512 entries</td>
</tr>
<tr>
<td></td>
<td>4-way, LRU replacement</td>
<td>L2 D-TLB: 512 entries</td>
</tr>
<tr>
<td>TLB misses</td>
<td>Handled in hardware</td>
<td>Handled in hardware</td>
</tr>
</tbody>
</table>
Memory Protection

- Different tasks can share parts of their virtual address spaces
  - But need to protect against errant access
  - Requires OS assistance

- Hardware support for OS protection
  - Privileged supervisor mode (aka kernel mode)
  - Privileged instructions
  - Page tables and other state information only accessible in supervisor mode
  - System call exception (e.g., syscall in MIPS)

---

Sharing

- Shared physical pages through mappings
- This raises issues with the cache
  - Synonym problem: we will not address that here
Disk Storage

- Nonvolatile, rotating magnetic storage

![Disk Drive Terminology Diagram]

- Data is recorded on concentric tracks on both sides of a platter
  - Tracks are organized as fixed size (bytes) sectors
- Corresponding tracks on all platters form a cylinder
- Data is addressed by three coordinates: cylinder, platter, and sector
Disk Sectors and Access

- Each sector records
  - Sector ID
  - Data (512 bytes, 4096 bytes proposed)
  - Error correcting code (ECC)
    - Used to hide defects and recording errors
  - Synchronization fields and gaps

- Access to a sector involves
  - Queuing delay if other accesses are pending
  - Seek: move the heads
  - Rotational latency
  - Data transfer
  - Controller overhead

Disk Performance

- Actuator moves (seek) the correct read/write head over the correct sector
  - Under the control of the controller

- Disk latency = controller overhead + seek time + rotational delay + transfer delay
  - Seek time and rotational delay are limited by mechanical parts
Disk Performance

- Seek time determined by the current position of the head, i.e., what track is it covering, and the new position of the head
  - milliseconds

- Average rotational delay is time for 0.5 revolutions

- Transfer rate is a function of bit density

---

Disk Access Example

- Given
  - 512B sector, 15,000rpm, 4ms average seek time, 100MB/s transfer rate, 0.2ms controller overhead, idle disk

- Average read time
  - 4ms seek time
    - $\frac{1}{2} / (15,000/60) = 2$ms rotational latency
    - $512 / 100$MB/s = 0.005ms transfer time
    - 0.2ms controller delay
    - = 6.2ms

- If actual average seek time is 1ms
  - Average read time = 3.2ms
Disk Performance Issues

- Manufacturers quote average seek time
  - Based on all possible seeks
  - Locality and OS scheduling lead to smaller actual average seek times

- Smart disk controller allocate physical sectors on disk
  - Present logical sector interface to host
  - Standards: SCSI, ATA, SATA

- Disk drives include caches
  - Prefetch sectors in anticipation of access
  - Avoid seek and rotational delay
  - Maintain caches in host DRAM

Arrays of Inexpensive Disks: Throughput

- Data is striped across all disks
- Visible performance overhead of drive mechanics is amortized across multiple accesses
- Scientific workloads are well suited to such organizations
Arrays of Inexpensive Disks: Request Rate

- Consider multiple read requests for small blocks of data
- Several I/O requests can be serviced concurrently

Reliability of Disk Arrays

- The reliability of an array of N disks is lower than the reliability of a single disk
  - Any single disk failure will cause the array to fail
  - The array is N times more likely to fail
- Use redundant disks to recover from failures
  - Similar to use of error correcting codes
- Overhead
  - Bandwidth and cost
RAID

- Redundant Array of Inexpensive (Independent) Disks
  - Use multiple smaller disks (c.f. one large disk)
  - Parallelism improves performance
  - Plus extra disk(s) for redundant data storage
- Provides fault tolerant storage system
  - Especially if failed disks can be “hot swapped”

RAID Level 0

- RAID 0 corresponds to use of striping with no redundancy
- Provides the highest performance
- Provides the lowest reliability
- Frequently used in scientific and supercomputing applications where data throughput is important
RAID Level 1

- The disk array is “mirrored” or “shadowed” in its entirety
- Reads can be optimized
  - Pick the array with smaller queuing and seek times
- Performance sacrifice on writes – to both arrays

RAID 3: Bit-Interleaved Parity

- N + 1 disks
  - Data striped across N disks at byte level
  - Redundant disk stores parity
  - Read access
    - Read all disks
  - Write access
    - Generate new parity and update all disks
  - On failure
    - Use parity to reconstruct missing data
- Not widely used
**RAID Level 4: N+1 Disks**

- Data is interleaved in blocks, referred to as the striping unit and striping width
- Small reads can access subset of the disks
- A write to a single disk requires 4 accesses
  - read old block, write new block, read and write parity disk
- Parity disk can become a bottleneck

---

**The Small Write Problem**

- Two disk read operations followed by two disk write operations
### RAID 5: Distributed Parity

- **N + 1 disks**
  - Like RAID 4, but parity blocks distributed across disks
  - Avoids parity disk being a bottleneck

- **Widely used**

![Diagram of RAID 4 and RAID 5]

### RAID Summary

- RAID can improve performance and availability
  - High availability requires hot swapping

- Assumes independent disk failures
  - Too bad if the building burns down!

- See “Hard Disk Performance, Quality and Reliability”
Flash Storage

- Nonvolatile semiconductor storage
  - 100× – 1000× faster than disk
  - Smaller, lower power, more robust
  - But more $/GB (between disk and DRAM)

Flash Types

- NOR flash: bit cell like a NOR gate
  - Random read/write access
  - Used for instruction memory in embedded systems

- NAND flash: bit cell like a NAND gate
  - Denser (bits/area), but block-at-a-time access
  - Cheaper per GB
  - Used for USB keys, media storage, ...

- Flash bits wears out after 1000’s of accesses
  - Not suitable for direct RAM or disk replacement
  - Wear leveling: remap data to less used blocks
Solid State Disks

- Replace mechanical drives with solid state drives
- Superior access performance
- Adding another level to the memory hierarchy
  - Disk is the new tape!
- Wear-leveling management

The Memory Hierarchy

**The BIG Picture**

- Common principles apply at all levels of the memory hierarchy
  - Based on notions of caching
- At each level in the hierarchy
  - Block placement
  - Finding a block
  - Replacement on a miss
  - Write policy
Concluding Remarks

- Fast memories are small, large memories are slow
  - We really want fast, large memories 😊
  - Caching gives this illusion 😊

- Principle of locality
  - Programs use a small part of their memory space frequently

- Memory hierarchy
  - L1 cache ↔ L2 cache ↔ ... ↔ DRAM memory ↔ disk

- Memory system design is critical for multiprocessors

Study Guide

- Be able to trace through the page table and cache data structures on a memory reference (see sample problems)

- Understand how to allocate virtual pages to page frames to minimize conflicts in the cache

- Relationships between address translation, page size, and cache size.
  - For example, given a memory system design (page sizes, virtual and physical address spaces, cache parameters) understand the address breakdowns at different levels of the memory hierarchy

- Be able to map lines in a page to sets in the cache (identify the set from the address)
Study Guide

• Given a cache design and virtual address space and page size, define the pages (by their addresses) that may conflict in the cache

• Distinguish between a TLB miss, a data cache miss, and a page fault

Glossary

• Page Table
• Page Table Entry (PTE)
• Page fault
• Physical address
• Physical page
• Physically tagged cache

• Synonym problem
• Translation lookaside buffer (TLB)
• Virtual address
• Virtual page
• Virtually tagged cache