Guerrilla 7: Virtual Memory, ECC, IO

August 4, 2016
Virtual Memory

ECC, IO

Disk, Dependency
Why do we need Virtual Memory?

- Give each program the illusion that it has its own private address space.
- Provide protection between processes.
- Add disk to memory hierarchy.
Why do we need Virtual Memory?

- Give each program the illusion that it has its own **private address space**.
- Provide **protection** between processes.
- Add **disk** to memory hierarchy.
Solution: Paged Memory System

- Program’s view of memory can be broken up into pages
Solution: Paged Memory System

- Program’s view of memory can be broken up into pages

| page number | offset |
Solution: Paged Memory System

- Program’s view of memory can be broken up into pages
- Offset bits = \( \log_2(\text{Page Size}) \)
- VPN = VA bits - Offset bits
- PPN = PA bits - Offset bits
- Virtual Memory could possibly be greater than Physical Memory
From VPN To PPN: Page Table

- Allow storage of program pages non-contiguously!
- PT resides in main memory: too large to keep in register.
From VPN To PPN: Page Table

- Allow storage of program pages non-contiguously!
- PT resides in main memory: too large to keep in register.

- Each program has a page table
- Page table contains an entry for each program page
From VPN To PPN: Page Table

- Contains mapping from VA to PA.
From VPN To PPN: Page Table

- Contains mapping from VA to PA.

**Page Table Layout**

1) Index into PT using VPN
2) Check Valid and Access Rights bits
3) Combine PPN and offset
4) Use PA to access memory
From VPN To PPN: Page Table

- Contains mapping from VA to PA.

**Page Table Layout**

1) Index into PT using VPN

2) Check Valid and Access Rights bits

3) Combine PPN and offset

4) Use PA to access memory

What is the page is invalid? What if the user does not have right to access a certain page?
PTE: Page Table Entry

- Contains either PPN or indication not in main memory
- **Valid** = Valid page table entry
  - 1 → virtual page is in physical memory
  - 0 → OS needs to fetch page from disk
- **Access Rights** checked on every access to see if allowed (provides protection)
  - *Read*
  - *Write*
  - *Executable*: Can fetch instructions from page
Speedup: TLB

- A cache for Page Table
- Usually small, 32-128 entries
Overview
Virtual Memory
ECC, IO
Disk, Dependency

Speedup: TLB

- A cache for Page Table
- Usually small, 32-128 entries
- Usually fully associative cache. Why?
Speedup: TLB

- A cache for Page Table
- Usually small, 32-128 entries
- Usually fully associative cache. Why?
- TLB Reach: Amount of virtual address space that can be simultaneously mapped by TLB
- TLB Reach = number of TLB entries * page size
Address Translation using TLB

Overview
Virtual Memory
ECC, IO
Disk, Dependency

Guerrilla 7: Virtual Memory, ECC, IO
Address Translation walkthrough

Virtual Address

TLB Lookup

TLB Miss

Page Table “Walk”

Page not in Mem

Page Fault (OS loads page)

Find in Disk

Access Denied

SEGFAULT

Update TLB

Find in Mem

Access Permitted

Physical Address

Check cache

Protection Fault

Protection Check

TLB Hit
Oh I have a Page Fault

• Load the page off the disk into a free page of memory
  – Switch to some other process while we wait
• Interrupt thrown when page loaded and the process’ page table is updated
  – When we switch back to the task, the desired data will be in memory
• If memory full, replace page (LRU), writing back if necessary, and update both page table entries
  – Continuous swapping between disk and memory called “thrashing”
ECC, IO
Error Detection/Correction Code

- Memory system generates errors - accidentally flipped-bits)
Memory system generates errors - accidentally flipped-bits.

Extra bits are added to each M-bit data chunk to produce an N-bit "code word".

Those extra bits are a function of the real data.
Single-Error Correction: Hamming ECC

- Use *extra parity bits* to allow the position identification of a single error
  - Interleave parity bits *within* bits of data to form code word
  - **Note:** Number bits starting at 1 from the left
1) Use *all* bit positions in the code word that are powers of 2 for parity bits (1, 2, 4, 8, 16, …)
2) **All other bit positions** are for the data bits (3, 5, 6, 7, 9, 10, …)
**Single-Error Correction: Hamming ECC**

<table>
<thead>
<tr>
<th>Bit position</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Encoded data bits</strong></td>
<td>p1</td>
<td>p2</td>
<td>d1</td>
<td>p4</td>
<td>d2</td>
<td>d3</td>
<td>d4</td>
<td>p8</td>
<td>d5</td>
<td>d6</td>
<td>d7</td>
<td>d8</td>
<td>d9</td>
<td>d10</td>
<td>d11</td>
</tr>
<tr>
<td><strong>Parity bit coverage</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The table shows how Hamming ECC encodes data and parity bits for error correction. Each parity bit covers a set of data bits, allowing for the detection and correction of single-bit errors.
Memory-mapped IO

Program accesses device with control and data register.
Polling

CPU waits until the device is ready

Little overhead

Good for regular, frequent accesses
Polling

- CPU waits until the device is ready
- Little overhead
- Good for regular, frequent accesses
Interrupt

Device sends interrupt to processor, processor halts program and goes to interrupt handler.

▶ Large overhead - need to save program state
▶ Handler is inaccessible to user program
▶ Good for infrequent accesses
Interrupt

- Device sends interrupt to processor, processor halts program and goes to interrupt handler
- Large overhead - need to save program state
- Handler is inaccessible to user program
- Good for infrequent accesses
Dependency and RAID
Dependability Measures

- **Reliability**: Mean Time To Failure (MTTF)
- **Service interruption**: Mean Time To Repair (MTTR)
- **Mean Time Between Failures (MTBF)**
  - $MTBF = MTTR + MTTF$
- **Availability** = $MTTF / (MTTF + MTTR) = MTTF / MTBF$
- **Improving Availability**
  - Increase MTTF: more reliable HW/SW + fault tolerance
  - Reduce MTTR: improved tools and processes for diagnosis and repair
RAID Summary

- RAID 0: No redundancy
- RAID 1: Disk Mirroring/Shadowing
- RAID 2-4: Data Striping + Parity. (bit-striping, byte-striping, block striping)
- RAID 5: Interleaved parity.
RAID 1
Mirroring (no parity, no striping)
Writes to 2 disks
RAID 3

Byte-striping, spindles synchronized, dedicated parity disk

Reads multiple disks, writes multiple disks
RAID 4

Block-striping, dedicated parity disk
Reads 1 disk, writes multiple disks
Multiple reads possible; writes issue reads
RAID 5
Block-striping, striped parity
Reads 1 disk, writes multiple disks
Multiple reads & writes in parallel possible
Disk Access Time

Seek Time + Rotation Time + Transfer Time + Controller Processing Time
Disk Access Time

Seek Time + Rotation Time + Transfer Time + Controller Processing Time

Seek Time = avg # tracks moved * time to move track
= total # tracks/3 * time to move track
Disk Access Time

Seek Time + Rotation Time + Transfer Time + Controller Processing Time

Rotation Time = avg rotation * 1 full rotation time
= ½ * 1 full rotation time
Disk Access Time

Seek Time + Rotation Time + Transfer Time + Controller Processing Time

Transfer Time = Size / Transfer rate
Ending