Last Class

1. Page Faults
   - What They Are
   - How They Occur
   - Information They Provide
     - Faulting Address (x:cr2)
     - Why the fault occurred

2. Uses
   1. Copy-On-Write

Today

Uses: Demand Paging

Goal

[Diagram showing virtual address, physical memory, and disk with labels and arrows indicating the process of making it run]
Why?

1. Loading Executables
2. MMAP(2)
3. Using More Memory Than Physically Available

How?

1. PF Occurs
   - Check if page should be loaded from disk
2. Allocate Physical Page
   - Check if free physical pages are available
   - No Evict to free physical pages
Eviction

Today's Question: What page to evict?

Page Replacement Policy

Aside: Does this even matter anymore?

- CDNs
- Databases
**Policies**

- **FIFO** - First In First Out

- **Min (Optimal)** - Clairvoyant (knows the future)
  - Evict page that won't be accessed for the longest time

**Example**

3 physical pages

Access: ABC ABD ADB CB

FIFO:

- P0: A B C A B D A D B C B
- P1: B C h A
- P2: C B h

# of Page-Ins/Misses: 7

# of Hits: 4
Why It Matters

Avg Memory Access Time = \( p \) Page fault time + \((1-p)\) memory access latency

Assumption Memory Access Lat. \( \ll \) Page Fault Time
\( \sim 100\text{ns} \)
\( \sim 10\text{ms} = 10^7\text{ns} \)

LRU: Least Recently Used

\(\text{LRU: } A \ B \ C \ A \ B \ D \ A \ D \ B \ C \ B \)

\(\text{PO: } A \ h \ h \ h \ C \)

\(\text{P1: } B \ h \ h \ h \ h \ h \)

\(\text{P2: } C \ D \ h \ h \)

\# of Page-Ins / Misses \( 5 \)
\# of Hits \( 6 \)
LRU A B C D A B C D A B C D
P_0 A
P_1 B A D
P_2 C B
Min
P_0 A hh
P_1 B hh
P_2 C D hh

Trade-Offs

- FIFO + Simple
  - High miss rates

- Opt + Optimal
  - Know the future

- Clock + Empirically close to Min

Approximating LRU - Clock

Reference Bit (A)
**Miscellaneous**

- **Beladi's Anomaly**

  L7 FIFO

  \[ ABCDABEABCD \]

  \[ P_0 \]

  \[ P_1 \]

  \[ P_2 \]

  \[ P_3 \]

  3 Hit \quad 4 Hit
Thrashing

Page Faults for Accounting
Core i7 Page Table Translation

Virtual address

Offset into physical and virtual page

Physical address

VPN 1
VPN 2
VPN 3
VPN 4
VPO

L1 PT
Page global directory

L2 PT
Page upper directory

L3 PT
Page middle directory

L4 PT
Page table

L1 PTE
L2 PTE
L3 PTE
L4 PTE

40

40

40

40

Virtual page

Physical page

CR3
Physical address of L1 PT

512 GB region per entry

1 GB region per entry

2 MB region per entry

4 KB region per entry

PPN

PPO

40

12

125 GB region per entry

1 GB region per entry

2 MB region per entry

4 KB region per entry

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition
Core i7 Level 4 Page Table Entries

Each entry references a 4K child page. Significant fields:

**P**: Child page is present in memory (1) or not (0)

**R/W**: Read-only or read-write access permission for this page

**U/S**: User or supervisor mode access

**WT**: Write-through or write-back cache policy for this page

**A**: Reference bit (set by MMU on reads and writes, cleared by software)

**D**: Dirty bit (set by MMU on writes, cleared by software)

**Page physical base address**: 40 most significant bits of physical page address (forces pages to be 4KB aligned)

**XD**: Disable or enable instruction fetches from this page.
### Core i7 Level 1-3 Page Table Entries

<table>
<thead>
<tr>
<th>XD</th>
<th>Unused</th>
<th>Page table physical base address</th>
<th>Unused</th>
<th>G</th>
<th>PS</th>
<th>A</th>
<th>CD</th>
<th>WT</th>
<th>U/S</th>
<th>R/W</th>
<th>P=1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>G</td>
<td>PS</td>
<td>A</td>
<td>CD</td>
<td>WT</td>
<td>U/S</td>
<td>R/W</td>
<td>P=0</td>
</tr>
</tbody>
</table>

#### Each entry references a 4K child page table. Significant fields:

- **P**: Child page table present in physical memory (1) or not (0).
- **R/W**: Read-only or read-write access access permission for all reachable pages.
- **U/S**: user or supervisor (kernel) mode access permission for all reachable pages.
- **WT**: Write-through or write-back cache policy for the child page table.
- **A**: Reference bit (set by MMU on reads and writes, cleared by software).
- **PS**: Page size: if bit set, we have 2 MB or 1 GB pages (bit can be set in Level 2 and 3 PTEs only).

**Page table physical base address**: 40 most significant bits of physical page table address (forces page tables to be 4KB aligned)

**XD**: Disable or enable instruction fetches from all pages reachable from this PTE.
<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>0: The fault was caused by a non-present page.</td>
</tr>
<tr>
<td></td>
<td>1: The fault was caused by a page-level protection violation.</td>
</tr>
<tr>
<td>W/R</td>
<td>0: The access causing the fault was a read.</td>
</tr>
<tr>
<td></td>
<td>1: The access causing the fault was a write.</td>
</tr>
<tr>
<td>U/S</td>
<td>0: A supervisor-mode access caused the fault.</td>
</tr>
<tr>
<td></td>
<td>1: A user-mode access caused the fault.</td>
</tr>
<tr>
<td>RSVD</td>
<td>0: The fault was not caused by reserved bit violation.</td>
</tr>
<tr>
<td></td>
<td>1: The fault was caused by a reserved bit set to 1 in some paging-structure entry.</td>
</tr>
<tr>
<td>I/D</td>
<td>0: The fault was not caused by an instruction fetch.</td>
</tr>
<tr>
<td></td>
<td>1: The fault was caused by an instruction fetch.</td>
</tr>
<tr>
<td>PK</td>
<td>0: The fault was not caused by protection keys.</td>
</tr>
<tr>
<td></td>
<td>1: There was a protection-key violation.</td>
</tr>
<tr>
<td>SGX</td>
<td>0: The fault is not related to SGX.</td>
</tr>
<tr>
<td></td>
<td>1: The fault resulted from violation of SGX-specific access-control requirements.</td>
</tr>
</tbody>
</table>

**Figure 4-12. Page-Fault Error Code**
End-to-end Core i7 Address Translation

CPU

Virtual address (VA)

VPN
VPO

TLBT TLBI

TLB hit

TLB miss

L1 TLB (16 sets, 4 entries/set)

VPN1 VPN2 VPN3 VPN4

9 9 9 9

CR3

Page tables

VPN1 VPN2 VPN3 VPN4

TLB

32/64

Result

L2, L3, and main memory

L1 hit

L1 miss

L1 d-cache
(64 sets, 8 lines/set)

Physical address (PA)

CT CI CO

32

12

40

12

40

6

6

PTE

PTE

PTE

PTE

PTE

PTE

PTE

PTE

PTE
Cute Trick for Speeding Up L1 Access

Observation

- Bits that determine CI identical in virtual and physical address
- Can index into cache while address translation taking place
- Cache carefully sized to make this possible: 64 sets, 64-byte cache blocks
- Means 6 bits for cache index, 6 for cache offset
- That’s 12 bits; matches $VPO, PPO \rightarrow$ One reason pages are $2^{12}$ bits = 4 KB
Virtual Address Space of a Linux Process

- **Process-specific data structs** (ptables, task and mm structs, kernel stack)
- **Physical memory**
- **Kernel code and data**

- **User stack**
- **Memory mapped region for shared libraries**
- **Runtime heap** (malloc)
- **Uninitialized data (.bss)**
- **Initialized data (.data)**
- **Program text (.text)**

- **Kernel virtual memory**
- **Process virtual memory**

**Identical for each process**
```c
void exception(int64_t* reg) {
    // Copy the saved registers into the 'current' process descriptor
    current->g Registers = reg;
    set_pagerable(kernel_pageetable);

    // It can be useful to log events using 'log_print'.
    // Events logged this way are stored in the host's 'log.txt' file.
    log_print("proc %x exception %x", current->psd, reg->reg_table);

    // Show the current cursor location and memory state
    // (unless this is a kernel fault).
    if (reg->reg_table == INT_PAGEFAULT || (reg->reg_table & PFERR_USER)) {
        check_virtual_memory();
        memshow_phystbl();
        memshow_virtual();
    }

    #if TICKLIMIT
    if (tick == TICKLIMIT) {
        poweroff();
    }
```