CS202: VIRTUAL MEMORY

- LAST CLASS
  - WALKING THE PAGE TABLES

| 47 | 9 | 9 | 9 |
| L1Idx | L2Idx | L3Idx | L4Idx | Off |

- TLB: TRANSLATION LOOK ASIDE BUFFER
  - CACHE PAST TRANSLATION RESULTS
  - SAVE TIME!

TODAY
- WEENSY OS (LAB4)
- PAGE FAULTS & THEIR USES.

WEENSY OS
- START EARLY <- DON'T
- REVIEW SESSION: #61 7:15 PM 10/31
- GOAL HERE: PROVIDE SOME CONTEXT.

P-foonk.c

P-foonk | P-alloc ...

WEENSYOS KERNEL

P-foonk.c

KERNEL S1.c

AMD64
PROCESS CONTROL BLOCK (PCB)

```
struct proc * (kernel.h)
```

```
x86-64-page-table* p-page-table
```

![Image of PCB structure]

What you will often be manipulating

**Metadata to Track Physical Pages**

```
physical-pageinfo
```

```
paginof (kernel.c)
```

```
sys-page-alloc
```

```
owner
```

```
FREE RESERVED KERNEL
```

```
> refcnt -> How many page tables map this page.
```

```
Manipulated by assign-physical-page.
```

**Manipulating Virtual Addresses**

- From Userspace:

```
sys-page-alloc (void* addr) [process.h]
```

```
-> Allocate a page at VA addr
```
From kernel:

virtual-memory-map

\[ \begin{array}{c}
\text{PAGE ALIGNED} \\
\hline
1000 \to 0 \times hff \\
/va \to 0xff \\
\text{Goal: Map}
\end{array} \]

\[ \begin{array}{c}
\text{va} \\
\text{va+sz} \\
\Rightarrow \\
\text{pa} \\
\text{pa+sz}
\end{array} \]

Assumption: Caller has already checked \([\text{pa} \ldots \text{pa+sz}]\) can be safely mapped.

How is the allocator used?

To mapped for kernel
Q: Want to map a page at virtual address 0x401000 (VPN 1025)?

\[ \text{movq } 0x401000, \%rax \]
Processor throws an exception (trap)

Stack

Stack when exception occurred

trap frame

CR2 → address which caused fault (0x401068)

%rip → trap handler (in kernel)

%rip → trap handler (in kernel)

How used

1. Demand paging to overcommit memory
   (next class)

2. Copy on write.
CORE IDEA
- Delay actually copying until necessary
- When necessary?
  ON MODIFICATION

ACCOUNTING
Linux VA Layout
Core i7 Page Table Translation

Virtual address

Offset into physical and virtual page

Physical address

Physical address of 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

PPN

PPO

512 GB region per entry

1 GB region per entry

2 MB region per entry

4 KB region per entry

CR3
Physical address of L1 PT

40

40

40

40

40

40
## Core i7 Level 4 Page Table Entries

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>52</th>
<th>51</th>
<th>12</th>
<th>11</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>XD</td>
<td>Unused</td>
<td>Page physical base address</td>
<td>Unused</td>
<td>G</td>
<td>D</td>
<td>A</td>
<td>CD</td>
<td>WT</td>
<td>U/S</td>
<td>R/W</td>
<td>P=1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Available for OS (for example, if page location on disk)  
P=0

### 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>63</th>
<th>62</th>
<th>52</th>
<th>51</th>
<th>12</th>
<th>11</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>XD</td>
<td>Unused</td>
<td>Page table physical base address</td>
<td>Unused</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=1</td>
<td></td>
<td></td>
<td></td>
<td></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>Field</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>0</td>
<td>The fault was caused by a non-present page.</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>The fault was caused by a page-level protection violation.</td>
</tr>
<tr>
<td>W/R</td>
<td>0</td>
<td>The access causing the fault was a read.</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>The access causing the fault was a write.</td>
</tr>
<tr>
<td>U/S</td>
<td>0</td>
<td>A supervisor-mode access caused the fault.</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>A user-mode access caused the fault.</td>
</tr>
<tr>
<td>RSVD</td>
<td>0</td>
<td>The fault was not caused by reserved bit violation.</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>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</td>
<td>The fault was not caused by an instruction fetch.</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>The fault was caused by an instruction fetch.</td>
</tr>
<tr>
<td>PK</td>
<td>0</td>
<td>The fault was not caused by protection keys.</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>There was a protection-key violation.</td>
</tr>
<tr>
<td>SGX</td>
<td>0</td>
<td>The fault is not related to SGX.</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>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

VPN1

VPN2

VPN3

VPN4

CR3

Page tables

TLB

hit

miss

L1 TLB (16 sets, 4 entries/set)

VPN1

VPN2

VPN3

VPN4

PTE

PTE

PTE

PTE

CR3

Page tables

Result

L1 hit

L1 miss

L2, L3, and main memory

Physical address (PA)

VPN1

VPN2

VPN3

VPN4

PTE

PTE

PTE

PTE

Page tables

CPU

36/12

TLB

32/4

VPN

VPO

...
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 → 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

Identical for each process

- 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

%rsp
brk
0x00400000