CS444/544
Operating Systems II

Lecture 5
Virtual Memory Layout
4/18/2023

Acknowledgement: Slides drawn heavily from Yeongjin Jiang
Due Dates

• Lab 1 Due passed
  • 75% if submitted by 4/24 11:59 PM
  • 50% if submitted by 6/9 11:59 PM

• Please tag your commit accurately
  • Tag name: lab1-final
  • How to delete local/remote tag: [https://devconnected.com/how-to-delete-local-and-remote-tags-on-git/](https://devconnected.com/how-to-delete-local-and-remote-tags-on-git/)
    • Delete local tag: `git tag -d lab1-final`
    • Delete remote tag: `git push --delete origin lab1-final`

• Extra credits for challenge
  • Printing colors on console when typing ‘show’ as the command: +1%
Recap: Synchronizing TLB with Page Table

• CPU uses the TLB for caching Page Table Entries

• What will happen if content in TLB mismatches to the PTE in the page table?
  • Access wrong physical memory...
  • Does not honor the correct privilege in PTE (if we updated PTE after caching)
  • Running a new process with new CR3
    • Use old process’s mapping, wrong access
### TLB and Page Table Update

**Address**

0x12349678

#### TLB

<table>
<thead>
<tr>
<th>VPN</th>
<th>PPN</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x12345</td>
<td>0x0</td>
<td>1</td>
</tr>
<tr>
<td>0x12346</td>
<td>0x5</td>
<td>1</td>
</tr>
<tr>
<td>0x12347</td>
<td>0xff</td>
<td>1</td>
</tr>
<tr>
<td>0x12348</td>
<td>0xffff</td>
<td>1</td>
</tr>
<tr>
<td>0x12349</td>
<td>0x101</td>
<td>1</td>
</tr>
</tbody>
</table>

#### Page Directory Entry

<table>
<thead>
<tr>
<th>Addr PT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>0x3ff</td>
</tr>
<tr>
<td>0x48</td>
</tr>
<tr>
<td>..</td>
</tr>
</tbody>
</table>

#### Page Table Entry

<table>
<thead>
<tr>
<th>Addr PT</th>
<th>0xff</th>
<th>0xffff</th>
<th>0x101</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x347</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x348</td>
<td>0xff</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x349</td>
<td>0x101</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### TLB and Page Table Update

#### TLB

<table>
<thead>
<tr>
<th>VPN</th>
<th>PPN</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x12345</td>
<td>0x0</td>
<td>1</td>
</tr>
<tr>
<td>0x12346</td>
<td>0x5</td>
<td>1</td>
</tr>
<tr>
<td>0x12347</td>
<td>0xff</td>
<td>1</td>
</tr>
<tr>
<td>0x12348</td>
<td>0xffff</td>
<td>1</td>
</tr>
<tr>
<td>0x12349</td>
<td>0x101</td>
<td>1</td>
</tr>
</tbody>
</table>

#### Page Directory Entry

<table>
<thead>
<tr>
<th>Addr PT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>..</td>
</tr>
<tr>
<td>0x48</td>
</tr>
<tr>
<td>0x3ff</td>
</tr>
</tbody>
</table>

#### Page Table Entry

<table>
<thead>
<tr>
<th>Addr PT</th>
<th>&quot;flag&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>&quot;flag&quot;</td>
</tr>
<tr>
<td>0x347</td>
<td>0xff</td>
</tr>
<tr>
<td>0x348</td>
<td>0xffff</td>
</tr>
<tr>
<td>0x349</td>
<td>0x102</td>
</tr>
</tbody>
</table>

---

### Address

0x12349678

---

**Update**:
TLB and Page Table Update

Address 0x12349678

We need to invalidate this entry

<table>
<thead>
<tr>
<th>VPN</th>
<th>PPN</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x12345</td>
<td>0x0</td>
<td>1</td>
</tr>
<tr>
<td>0x12346</td>
<td>0x5</td>
<td>1</td>
</tr>
<tr>
<td>0x12347</td>
<td>0xff</td>
<td>1</td>
</tr>
<tr>
<td>0x12348</td>
<td>0xffff</td>
<td>1</td>
</tr>
<tr>
<td>0x12349</td>
<td>0x101</td>
<td>0</td>
</tr>
</tbody>
</table>

Page Directory Entry

| 0 | Addr PT |
|..| Addr PT |
|0x48 | Addr PT |
|0x3ff | Addr PT |

Page Table Entry

| 0 | Addr PT |
|0x347 | 0xff | FLAG |
|0x348 | 0xffff | FLAG |
|0x349 | 0x102 | FLAG |

Update
TLB and Process Context Switch

<table>
<thead>
<tr>
<th>Address 0x12349678</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR3</td>
</tr>
</tbody>
</table>

### TLB

<table>
<thead>
<tr>
<th>VPN</th>
<th>PPN</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x12345</td>
<td>0x0</td>
<td>1</td>
</tr>
<tr>
<td>0x12346</td>
<td>0x5</td>
<td>1</td>
</tr>
<tr>
<td>0x12347</td>
<td>0xff</td>
<td>1</td>
</tr>
<tr>
<td>0x12348</td>
<td>0xffff</td>
<td>1</td>
</tr>
<tr>
<td>0x12349</td>
<td>0x101</td>
<td>1</td>
</tr>
</tbody>
</table>

### Page Directory Entry

<table>
<thead>
<tr>
<th>Address</th>
<th>Addr PT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Addr PT</td>
</tr>
<tr>
<td>..</td>
<td>Addr PT</td>
</tr>
<tr>
<td>0x48</td>
<td>Addr PT</td>
</tr>
<tr>
<td>0x3ff</td>
<td>Addr PT</td>
</tr>
</tbody>
</table>

### Page Table Entry

<table>
<thead>
<tr>
<th>Addr PT</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Addr PT</td>
</tr>
<tr>
<td>0x347</td>
<td>0xff</td>
</tr>
<tr>
<td>0x348</td>
<td>0xffff</td>
</tr>
<tr>
<td>0x349</td>
<td>0x101</td>
</tr>
</tbody>
</table>
TLB and Process Context Switch

Address 0x12349678

CR3

TLB

<table>
<thead>
<tr>
<th>VPN</th>
<th>PPN</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x12345</td>
<td>0x0</td>
<td>1</td>
</tr>
<tr>
<td>0x12346</td>
<td>0x5</td>
<td>1</td>
</tr>
<tr>
<td>0x12347</td>
<td>0xff</td>
<td>1</td>
</tr>
<tr>
<td>0x12348</td>
<td>0xfff</td>
<td>1</td>
</tr>
<tr>
<td>0x12349</td>
<td>0x101</td>
<td>1</td>
</tr>
</tbody>
</table>

Page Directory Entry

<table>
<thead>
<tr>
<th>Addr PT</th>
<th>0</th>
<th>Addr PT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x48</td>
<td>0</td>
<td>Addr PT</td>
</tr>
<tr>
<td>0x3ff</td>
<td>0</td>
<td>Addr PT</td>
</tr>
<tr>
<td>0x348</td>
<td>0xff</td>
<td>FLAG</td>
</tr>
<tr>
<td>0x347</td>
<td>0xff</td>
<td>FLAG</td>
</tr>
</tbody>
</table>

Page Table Entry

<table>
<thead>
<tr>
<th>Addr PT</th>
<th>0</th>
<th>Addr PT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x349</td>
<td>0x101</td>
<td>FLAG</td>
</tr>
<tr>
<td>0x348</td>
<td>0xfff</td>
<td>FLAG</td>
</tr>
<tr>
<td>0x347</td>
<td>0x20</td>
<td>FLAG</td>
</tr>
<tr>
<td>0x346</td>
<td>0x30</td>
<td>FLAG</td>
</tr>
<tr>
<td>0x345</td>
<td>0x50</td>
<td>FLAG</td>
</tr>
</tbody>
</table>
TLB and Process Context Switch

We need to invalidate all previous entries..

TLB

<table>
<thead>
<tr>
<th>VPN</th>
<th>PPN</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x12345</td>
<td>0x0</td>
<td>0</td>
</tr>
<tr>
<td>0x12346</td>
<td>0x5</td>
<td>0</td>
</tr>
<tr>
<td>0x12347</td>
<td>0xff</td>
<td>0</td>
</tr>
<tr>
<td>0x12348</td>
<td>0xffff</td>
<td>0</td>
</tr>
<tr>
<td>0x12349</td>
<td>0x101</td>
<td>0</td>
</tr>
</tbody>
</table>

Address 0x12349678

CR3
Updating Page Table

• When updating a Page Table Entry
  • We must invalidate TLB for that entry
  • invlpg

**INVLPGL — Invalidate TLB Entries**

<table>
<thead>
<tr>
<th>Op/En</th>
<th>64-Bit Mode</th>
<th>Compat/Leg Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>Valid</td>
<td>Valid</td>
<td>Invalidate TLB entries for page containing ( m ).</td>
</tr>
</tbody>
</table>
Updating Page Table

• In JOS (kern/pmap.c & inc/x86.h)

```c
// Invalidate a TLB entry, but only if the page tables being
// edited are the ones currently in use by the processor.

void
tlb_invalidate(pde_t *pgdir, void *va)
{
    // Flush the entry only if we're modifying the current address space.
    // For now, there is only one address space, so always invalidate.
    invlpg(va);
}

static inline void
invlpg(void *addr)
{
    asm volatile("invlpg (%0)" : : "r" (addr) : "memory");
}
```
Intel 32-bit Processor uses a 2-level page table

- Virtual address
- Page directory (level 1) 
- Page table (level 2)
- Physical page!
Recap – Page Table & Addr Translation

Virtual | Physical
---|---
0x8048000 | 0x10000
0x8049000 | 0x11000
0x804a000 | 0x50000

CR3[0x20]

Mem access #1

CR3

Page Directory Entry
0 | Addr PT
.. | Addr PT
0x20 | Addr PT
0x3ff | Addr PT

Page Table Entry
0 | Addr PT
0 | Addr PT
0x48 | 0x10000
0x49 | 0x11000
0x4a | 0x50000

Mem access #2

Mem access #3 (required)
Today’s Topic

• Page Permissions

• Virtual Memory Layout

• How JOS Manages Physical Memory?
Page Directory / Table Entry (PDE/PTE)

• Top 20 bits: physical page number
  • Physical page number of a page table (PDE)
  • Physical page number of the requested virtual address (PTE)

• Lower 12 bits: some flags
  • Permission
  • Etc.
Permission Flags

- **PTE_P (PRESENT)**
  - 0: invalid entry
  - 1: valid entry

- **PTE_W (WRITABLE)**
  - 0: read only
  - 1: writable

- **PTE_U (USER)**
  - 0: kernel (only ring 0 can access)
  - 1: user (accessible by ring 3)

<table>
<thead>
<tr>
<th>Addr</th>
<th>Page Table Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x48</td>
<td>0x10000 &lt;&lt; 12</td>
</tr>
<tr>
<td>0x49</td>
<td>0x11000 &lt;&lt; 12</td>
</tr>
<tr>
<td>0x4a</td>
<td>0x50000 &lt;&lt; 12</td>
</tr>
</tbody>
</table>
When does CPU check Permission Bits?

- In address translation
  - 1. Virtual address
    - Checks permission bits in PDE
  - 2. PDE = CR3[PDX]
    - Checks permission bits in PDE
  - 3. PTE = PDE[PTX]
    - Checks permission bits in PTE
CPU checks PDE permission first and then PTE permission next...

• A virtual memory address is inaccessible if PDE disallows the access

• A virtual memory address is inaccessible if PTE disallows the access

• Both PDE and PTE should allow the access...
PDE/PTE Permission Examples 0

• Virtual address 0x01020304

• PDE: PTE_P | PTE_W | PTE_U

• PTE: PTE_P | PTE_W | PTE_U

• **Valid, accessible by ring 3, and writable**

• PTE_P (PRESENT)
  • 0: invalid entry
  • 1: valid entry

• PTE_W (WRITABLE)
  • 0: read only
  • 1: writable

• PTE_U (USER)
  • 0: kernel (only ring 0 can access)
  • 1: user (accessible by ring 3)
PDE/PTE Permission Examples 1

• Virtual address 0x01020304

• PDE: PTE_P | PTE_W | PTE_U

• PTE: PTE_P | PTE_U

• Valid, accessible by ring 3, but not writable

• PTE_P (PRESENT)
  • 0: invalid entry
  • 1: valid entry

• PTE_W (WRITABLE)
  • 0: read only
  • 1: writable

• PTE_U (USER)
  • 0: kernel (only ring 0 can access)
  • 1: user (accessible by ring 3)
PDE/PTE Permission Examples 2

• Virtual address 0x01020304

• PDE: PTE_P | PTE_U

• PTE: PTE_P | PTE_W | PTE_U

• Valid, accessible by ring 3, but not writable

• PTE_P (PRESENT)
  • 0: invalid entry
  • 1: valid entry

• PTE_W (WRITABLE)
  • 0: read only
  • 1: writable

• PTE_U (USER)
  • 0: kernel (only ring 0 can access)
  • 1: user (accessible by ring 3)
PDE/PTE Permission Examples 3

• Virtual address 0x01020304

• PDE: PTE_P | PTE_W | PTE_U

• PTE: PTE_P

• valid, inaccessible by ring3, not writable

• PTE_P (PRESENT)
  • 0: invalid entry
  • 1: valid entry

• PTE_W (WRITABLE)
  • 0: read only
  • 1: writable

• PTE_U (USER)
  • 0: kernel (only ring 0 can access)
  • 1: user (accessible by ring 3)
PDE/PTE Permission Examples 4

• Virtual address 0x01020304

• PDE: PTE_P | PTE_W

• PTE: PTE_P | PTE_U

• valid, inaccessible by ring3, not writable

• PTE_P (PRESENT)
  • 0: invalid entry
  • 1: valid entry

• PTE_W (WRITABLE)
  • 0: read only
  • 1: writable

• PTE_U (USER)
  • 0: kernel (only ring 0 can access)
  • 1: user (accessible by ring 3)
PDE/PTE Permission Examples 5

- Virtual address 0x01020304
- PDE: PTE_P | PTE_U
- PTE: PTE_U
- invalid

- PTE_P (PRESENT)
  - 0: invalid entry
  - 1: valid entry
- PTE_W (WRITABLE)
  - 0: read only
  - 1: writable
- PTE_U (USER)
  - 0: kernel (only ring 0 can access)
  - 1: user (accessible by ring 3)
PDE/PTE Permission Examples 6

• Virtual address 0x01020304

• PDE: PTE_U

• PTE: PTE_P | PTE_U

• invalid

• PTE_P (PRESENT)
  • 0: invalid entry
  • 1: valid entry

• PTE_W (WRITABLE)
  • 0: read only
  • 1: writable

• PTE_U (USER)
  • 0: kernel (only ring 0 can access)
  • 1: user (accessible by ring 3)
Can you setup a page permission as...

- Kernel: R, User: --
  - PTE_P

- Kernel: R, User: R
  - PTE_P | PTE_U

- Kernel: RW, User: RW
  - PTE_P | PTE_U | PTE_W
You can’t setup a page permission as...

- Kernel: RW, User: R
  - PTE_P | PTE_W | PTE_U -> User RW...
  - PTE_P | PTE_U -> User --

- Kernel: R, User: RW
  - PTE_P | PTE_U | PTE_W -> Kernel RW...
  - PTE_P | PTE_U -> User R...

- Kernel: --, User: RW
  - PTE_P | PTE_U | PTE_W -> Kernel RW...
You can enable such a conflicting permission setup by having N-to-1 mapping

- Virtual to physical address mapping is in N-to-1 relation
  - N number of virtual addresses could be mapped to 1 physical address

- E.g., for a physical address 0x100000
  - JOS maps VA 0x100000 to PA 0x100000
  - JOS maps VA 0xf0100000 to PA 0x100000

- Why?
  - EIP before enabling paging: 0x100025
  - EIP after enabling paging: 0x100028
Sharing a Physical Page

• Example: Loading of the same program

Process 0, runs /bin/bash, loads at virt addr 0x35555000

<table>
<thead>
<tr>
<th>Page Table Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>0x155 0x10303</td>
</tr>
</tbody>
</table>

Process 1, runs /bin/bash, loads at virt addr 0x43132000

<table>
<thead>
<tr>
<th>Page Table Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>0x132 0x10303</td>
</tr>
</tbody>
</table>

2 or more mappings to 0x10303000 is possible!
You can’t setup a page permission as...

- **Kernel: RW, User: R**
  - VA 0x00001000 -> PA 0x50000, PTE_P | PTE_U (User R)
  - VA 0xf0050000 -> PA 0x50000, PTE_P | PTE_W (Kernel RW)

- **Kernel: R, User: RW**
  - VA 0x00002000 -> PA 0x60000, PTE_P | PTE_U | PTE_W (User RW)
  - VA 0xf0060000 -> PA 0x60000, PTE_P (Kernel R)

- **Kernel: --, User: RW**
  - VA 0x00003000 -> PA 0x70000, PTE_P | PTE_U | PTE_W
  - VA 0xf0070000 -> PA 0x70000, 0 for flag...
PDE/PTE Permissions CAVEAT

• A virtual address access is allowed if both PDE and PTE entries allows the access...

• General practice: put a more permissive permission bits in PDE, and be strict on setting permission bits in PTE

• For a conflicting permission setup for Kernel/User, add an additional virtual address mapping can enable such a setup.
How To Create a Page Directory?

• For a 32-bit Intel processor, we use only 1 page for a Page Directory

One page, 4KB

Each entry is 4-byte (32 bits)

4096 / 4 = 1024 entries

1024 == 2^{10}

10-bit index for PD
How To Create a Page Table?

• For a 32-bit Intel processor, we use only 1 page for a Page Table.

One page, 4KB

Each entry is 4-byte (32 bits)

4096 / 4 = 1024 entries

1024 == 2^{10}

10-bit index for PT
Intel 32-bit Processor uses a 2-level page table

• Virtual address

• Page directory (level 1)

• Page table (level 2)

• Physical page!
What will be the size of full PD/PT?

One page, 4KB

Each entry is 4-byte (32 bits)

4KB for page directory

4KB per each Page Table...
Size of Page Table

- 4 KB for a Page Directory (only one per each process)

- 1024 Page Tables available
  - 4 KB for a Page Table

- 4 KB (PD) + 1024 * 4 KB (PT)
  - 4 KB + 4MB
  - ~4MB, 4,198,400 bytes...
Virtual Memory Layout

- OS allocates a separate virtual memory space for each process
- Transparency
  - Do not have to worry about a system’s memory usage status
- Isolation
  - Others can’t access my virtual memory space

OS
0xc000000 ~ 0xffffffff
(1GB)

Stack

Libraries

heap

Program code/data
Virtual Memory Layout

• Each process will have almost the same mapping for the kernel but having a different mapping for user space

• Why?
  • Kernel is shared among processes
  • Each process could run different apps

Shared kernel mapping
Example: cat binary
Example: more binary

```
os2 ~/cs444/s21 337% more /proc/self/maps
00400000-00409000 r-xp 00000000 fd:02 872977
00608000-00609000 r--p 00008000 fd:02 872977
00609000-0060a000 rw-p 00009000 fd:02 872977
00f80000-00fa1000 rw-p 00000000 00:00 0
7f6d5f4e6000-7f6d65a28000 r--p 00000000 fd:02 51142142
7f6d65a28000-7f6d65bec000 r-xp 00000000 fd:02 16806708
7f6d65bec000-7f6d65deb000 ----p 001c4000 fd:02 16806708
7f6d65deb000-7f6d65def000 r--p 001c3000 fd:02 16806708
7f6d65def000-7f6d65df1000 rw-p 001c7000 fd:02 16806708
7f6d65df1000-7f6d65df6000 rw-p 00000000 00:00 0
7f6d65df6000-7f6d65e1b000 r-xp 00000000 fd:02 16806782
7f6d65e1b000-7f6d6601b000 ----p 00250000 fd:02 16806782
7f6d6601b000-7f6d6601f000 r--p 00250000 fd:02 16806782
7f6d6601f000-7f6d66020000 rw-p 00290000 fd:02 16806782
7f6d66020000-7f6d66042000 r-xp 00000000 fd:02 17757900
7f6d661ff000-7f6d66203000 rw-p 00000000 00:00 0
7f6d66237000-7f6d66238000 rw-p 00000000 00:00 0
7f6d66238000-7f6d6623f000 r--s 00000000 fd:02 1893394
7f6d6623f000-7f6d662410000 rw-p 00000000 00:00 0
7f6d66241000-7f6d66242000 r--p 00210000 fd:02 17757900
7f6d66242000-7f6d662430000 rw-p 00220000 fd:02 17757900
7f6d66243000-7f6d662440000 rw-p 00000000 00:00 0
7f7c06a30000-7f7c06a52000 rw-p 00000000 00:00 0
7f7c06b5000-7f7c06b5f000 r-xp 00000000 00:00 0
fffffffff600000-fffffffff601000 r-xp 00000000 00:00 0
```
```
Summary

• Page Directory Entry / Page Table Entry
  • Permission bits (P, W, U)
  • Permission: \{bits in PDE\} \cap \{bits in PTE\}

• Virtual memory is N-to-1 mapping
  • Sharing physical page
  • Allowing conflicting permission assignment
    • Kernel RW and User R

• Virtual Memory Layout
  • Shares kernel space (typically at the top of virtual memory space)
  • Can use user space arbitrarily (full transparency and isolation)