CS444/544
Operating Systems II

Lecture 5
Virtual Memory Layout
10/12/2023

Acknowledgement: Slides drawn heavily from Yeongjin Jiang
Due Dates

• Lab 1 Due: Monday 10/16 11:59 pm
  • commit before tag

• Extra credits for challenge
  • Printing colors on console when typing ‘show’ as the command: +1%
  • Test with “make qemu” instead of “make qemu-nox”
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

Directory Index (10-bits)
0x20

Table Index (10-bits)
0x48

Page number (20-bits)
0x08048

Offset (12-bits)
0x000

Mem access #1
CR3[0x20]

Mem access #2
PDE[0x48]

Mem access #3 (required)

Page Directory Entry
- 0: Addr PT
- ..: Addr PT
- 0x20: Addr PT
- 0x3ff: Addr PT

Page Table Entry
- 0: Addr PT
- 0x48: 0x10000
- 0x49: 0x11000
- 0x4a: 0x50000
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>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x48</td>
<td>0x10000 &lt;&lt; 12</td>
<td>PTE_U</td>
</tr>
<tr>
<td>0x49</td>
<td>0x110000 &lt;&lt; 12</td>
<td>PTE_P</td>
</tr>
<tr>
<td>0x4a</td>
<td>0x500000 &lt;&lt; 12</td>
<td>PTE_P</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]
  - 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_W -> 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

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

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

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

2 or more mappings to 0x10303000 is possible!

Physical memory

/bin/bash at Phys addr 0x10303000
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 allow 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)

<table>
<thead>
<tr>
<th>PDE 0</th>
<th>PDE 1</th>
<th>PDE 2</th>
<th>PDE 3</th>
<th>PDE ...</th>
<th>PDE ...</th>
<th>PDE 1022</th>
<th>PDE 1023</th>
</tr>
</thead>
</table>

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

PTE 0
PTE 1
PTE 2
PTE 3
PTE ...
PTE ...
PTE 1022
PTE 1023

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...
*What about amd64 (x86_64)?*

- Pointer/address size in 64-bit computer: 8 bytes
- Page size in 64-bit computer: still 4KB
- How many entries are there in each PD/PT?
  - Page size / pointer size => 4KB / 8B = 512 entries

Each entry is 8-byte (64 bits)
*We do not use the entire 64 bits memory space ...

• 32-bit memory space
  • $2^{32} = 4$GB
  • Lower 12 bits == offset
  • Only translate top 20 bits
    • 20 bits required to index all pages
  • 10 bits PDE, 10 bits PTE $\rightarrow$ 2 level page table

• 64-bit memory space
  • $2^{64} = 16$ EB == 16384 PB == 16777216 TB
  • Lower 12 bits == offset
  • Need to translate $64 - 12 = 52$ bits
  • Each table holds 512 entries, indexed by 9 bits
  • Need $52/9 \rightarrow 6$ level page table ...
*Use 48-bit address space instead

• Initial amd64 processors only use 48-bit virtual address space
  • $2^{48} = 256$ TB

• Each level: 512 entries → process 9 bits

• Lower 12 bits == offset
  • $48 - 12 = 36$
  • $36 / 9 = 4$

• Can be done using 4-level page table
Two-level Page Table (32 bit)

Linear address:
31 24 23 16 15 8 7 0

- CR3
- 32 bit PD entry
- page directory
- 32 bit PT entry
- page table
- 4K memory page

*) 32 bits aligned to a 4-KByte boundary
Four-level Page Table (64 bit)

FIGURE 5-20  x64 address translation.
*What’s more?

• 256 TB might not be enough...
  • E.g., analyzing online social network users in Facebook
    • More than 1 billion users, more than 1 trillion edges
    • 1 byte per edge = 1 TB

• 4 level == 48 bit
• Can be extended to 5 levels $\rightarrow$ $48 + 9 = 57$ bits
• 6 levels? Maybe, but $57 + 9 = 68$ bits $> 64$ bits
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
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

<table>
<thead>
<tr>
<th>Offset</th>
<th>Address</th>
<th>Permissions</th>
<th>Size</th>
<th>File Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>004000000-00409000</td>
<td>r-xp 00000000 fd:02 872977</td>
<td>/usr/bin/more</td>
<td></td>
<td></td>
</tr>
<tr>
<td>006000000-00609000</td>
<td>r--p 00000000 fd:02 872977</td>
<td>/usr/bin/more</td>
<td></td>
<td></td>
</tr>
<tr>
<td>006090000-0060a000</td>
<td>rw-p 00000000 fd:02 872977</td>
<td>/usr/bin/more</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00f800000-00fa1000</td>
<td>rw-p 00000000 00:00 0</td>
<td>[heap]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7f6d5f4e6000-7f6d65a28000</td>
<td>r--p 00000000 fd:02 51142142</td>
<td>/usr/lib/locale/locale-archive</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7f6d65a28000-7f6d65bec000</td>
<td>r-xp 00000000 fd:02 16806708</td>
<td>/usr/lib64/libc-2.17.so</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7f6d65bec000-7f6d65deb000</td>
<td>---p 001c4000 fd:02 16806708</td>
<td>/usr/lib64/libc-2.17.so</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7f6d65deb000-7f6d65def000</td>
<td>r--p 001c3000 fd:02 16806708</td>
<td>/usr/lib64/libc-2.17.so</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7f6d65def000-7f6d65df1000</td>
<td>rw-p 001c7000 fd:02 16806708</td>
<td>/usr/lib64/libc-2.17.so</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7f6d65df1000-7f6d65df6000</td>
<td>rw-p 00000000 00:00 0</td>
<td>/usr/lib64/libc-2.17.so</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7f6d65df6000-7f6d65e1b000</td>
<td>r-xp 00000000 fd:02 16806782</td>
<td>/usr/lib64/libc-2.17.so</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7f6d65e1b000-7f6d6601b000</td>
<td>---p 0025000 fd:02 16806782</td>
<td>/usr/lib64/libc-2.17.so</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7f6d6601b000-7f6d6601f000</td>
<td>r--p 0025000 fd:02 16806782</td>
<td>/usr/lib64/libc-2.17.so</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7f6d6601f000-7f6d66020000</td>
<td>rw-p 0029000 fd:02 16806782</td>
<td>/usr/lib64/libc-2.17.so</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7f6d66020000-7f6d66042000</td>
<td>r-xp 00000000 fd:02 17757900</td>
<td>/usr/lib64/libc-2.17.so</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7f6d66042000-7f6d6604f000</td>
<td>rw-p 00000000 00:00 0</td>
<td>/usr/lib64/libc-2.17.so</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7f6d6604f000-7f6d6613f000</td>
<td>rw-p 00000000 00:00 0</td>
<td>/usr/lib64/libc-2.17.so</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7f6d6613f000-7f6d66230000</td>
<td>rw-p 00000000 00:00 0</td>
<td>/usr/lib64/libc-2.17.so</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7f6d66230000-7f6d66238000</td>
<td>rw-p 00000000 00:00 0</td>
<td>/usr/lib64/libc-2.17.so</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7f6d66238000-7f6d6623f000</td>
<td>r--s 00000000 fd:02 1893394</td>
<td>/usr/lib64/libc-2.17.so</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7f6d6623f000-7f6d66241000</td>
<td>rw-p 00000000 00:00 0</td>
<td>/usr/lib64/libc-2.17.so</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7f6d66241000-7f6d66242000</td>
<td>rw-p 0021000 fd:02 17757900</td>
<td>/usr/lib64/libc-2.17.so</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7f6d66242000-7f6d66243000</td>
<td>rw-p 0022000 fd:02 17757900</td>
<td>/usr/lib64/libc-2.17.so</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7f6d66243000-7f6d66244000</td>
<td>rw-p 00000000 00:00 0</td>
<td>/usr/lib64/libc-2.17.so</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7ffe06a30000-7ffe06a52000</td>
<td>rw-p 00000000 00:00 0</td>
<td>/usr/lib64/libc-2.17.so</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7ffe06a52000-7ffe06b5f000</td>
<td>r-xp 00000000 00:00 0</td>
<td>/usr/lib64/libc-2.17.so</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fffffffff600000-fffffffffff601000 r-xp 00000000 00:00 0</td>
<td>/usr/lib64/libc-2.17.so</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[stack]
[vdso]
[vsyscall]
Summary

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