

15-213
 “The course that gives CMU its Zip!”

**Memory System
 Case Studies
 November 7, 2007**

Topics

- P6 address translation
- x86-64 extensions
- Linux memory management
- Linux page fault handling
- Memory mapping

class20.ppt

Intel P6

(Bob Colwell’s Chip, CMU Alumni)

Internal designation for successor to Pentium

- Which had internal designation P5

Fundamentally different from Pentium

- Out-of-order, superscalar operation

Resulting processors

- Pentium Pro (1996)
- Pentium II (1997)
 - L2 cache on same chip
- Pentium III (1999)
 - The freshwater fish machines

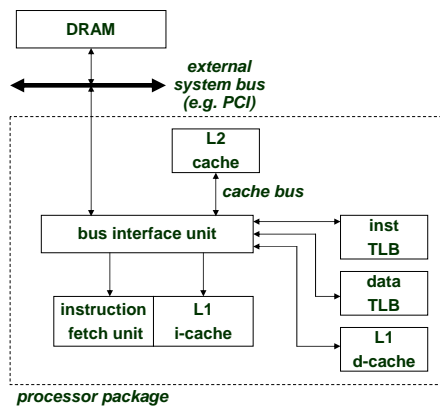
Saltwater fish machines: Pentium 4

- Different operation, but similar memory system
- Abandoned by Intel in 2005 for P6 based Core 2 Duo

- 2 -

15-213, F’07

P6 Memory System



32 bit address space

4 KB page size

L1, L2, and TLBs

- 4-way set associative

Inst TLB

- 32 entries
- 8 sets

Data TLB

- 64 entries
- 16 sets

L1 i-cache and d-cache

- 16 KB
- 32 B line size
- 128 sets

L2 cache

- unified
- 128 KB -- 2 MB

- 3 -

15-213, F’07

Review of Abbreviations

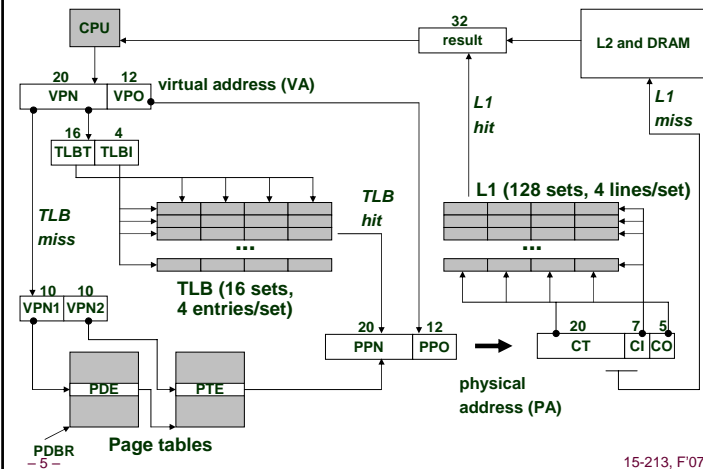
Symbols:

- Components of the virtual address (VA)
 - TLBI: TLB index
 - TLBT: TLB tag
 - VPO: virtual page offset
 - VPN: virtual page number
- Components of the physical address (PA)
 - PPO: physical page offset (same as VPO)
 - PPN: physical page number
 - CO: byte offset within cache line
 - CI: cache index
 - CT: cache tag

- 4 -

15-213, F’07

Overview of P6 Address Translation

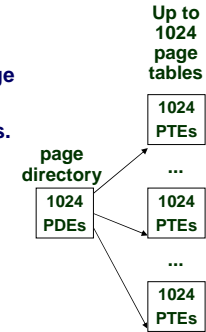


15-213, F'07

P6 2-level Page Table Structure

Page directory

- 1024 4-byte page directory entries (PDEs) that point to page tables
- One page directory per process.
- Page directory must be in memory when its process is running
- Always pointed to by PDBR



Page tables:

- 1024 4-byte page table entries (PTEs) that point to pages.
- Page tables can be paged in and out.

- 6 -

15-213, F'07

P6 Page Directory Entry (PDE)

31	12	11	9	8	7	6	5	4	3	2	1	0
Page table physical base addr		Avail	G	PS	A	CD	WT	U/S	R/W	P=1		

Page table physical base address: 20 most significant bits of physical page table address (forces page tables to be 4KB aligned)

Avail: These bits available for system programmers

G: global page (don't evict from TLB on task switch)

PS: page size 4K (0) or 4M (1)

A: accessed (set by MMU on reads and writes, cleared by software)

CD: cache disabled (1) or enabled (0)

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

U/S: user or supervisor mode access

R/W: read-only or read-write access

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

31	Available for OS (page table location in secondary storage)											1	0
												P=0	

- 7 -

15-213, F'07

P6 Page Table Entry (PTE)

31	12	11	9	8	7	6	5	4	3	2	1	0
Page physical base address		Avail	G	0	D	A	CD	WT	U/S	R/W	P=1	

Page base address: 20 most significant bits of physical page address (forces pages to be 4 KB aligned)

Avail: available for system programmers

G: global page (don't evict from TLB on task switch)

D: dirty (set by MMU on writes)

A: accessed (set by MMU on reads and writes)

CD: cache disabled or enabled

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

U/S: user/supervisor

R/W: read/write

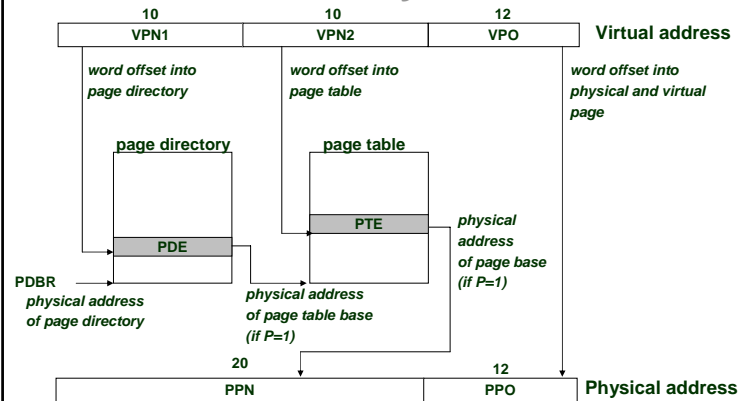
P: page is present in physical memory (1) or not (0)

31	Available for OS (page location in secondary storage)											1	0
												P=0	

- 8 -

15-213, F'07

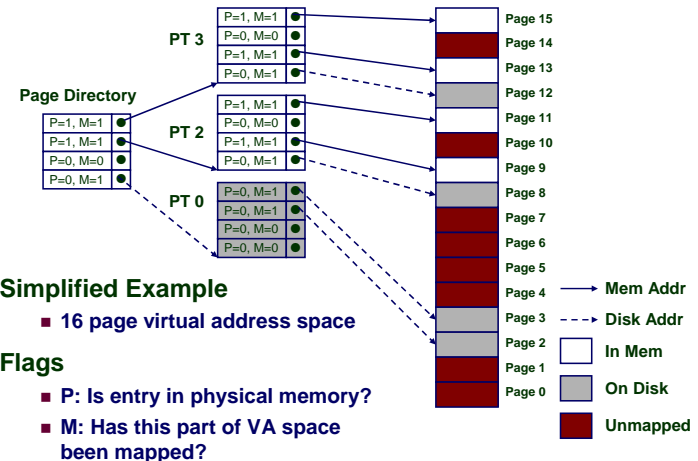
How P6 Page Tables Map Virtual Addresses to Physical Ones



- 9 -

15-213, F'07

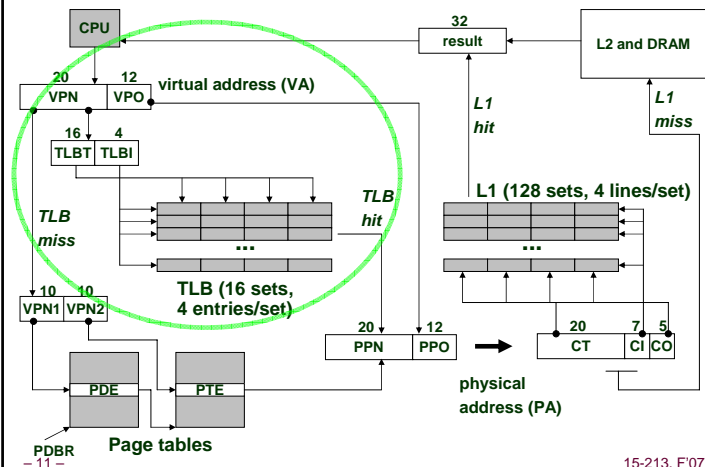
Representation of VM Address Space



- 10 -

15-213, F'07

P6 TLB Translation

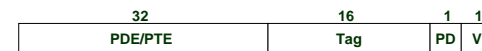


- 11 -

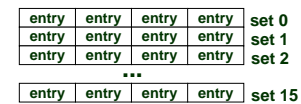
15-213, F'07

P6 TLB

TLB entry (not all documented, so this is speculative):



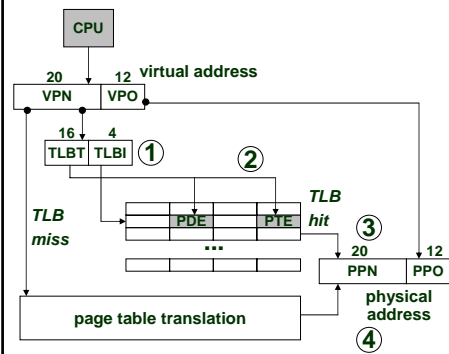
- V: indicates a valid (1) or invalid (0) TLB entry
- PD: is this entry a PDE (1) or a PTE (0)?
- tag: disambiguates entries cached in the same set
- PDE/PTE: page directory or page table entry
- Structure of the data TLB:
 - 16 sets, 4 entries/set



- 12 -

15-213, F'07

Translating with the P6 TLB

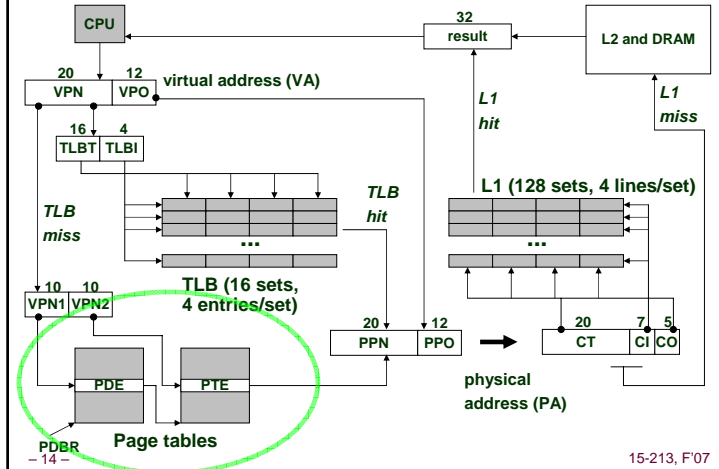


1. Partition VPN into TLBT and TLBI.
2. Is the PTE for VPN cached in set TLBI?
 - 3. Yes: then build physical address.
4. No: then read PTE (and PDE if not cached) from memory and build physical address.

- 13 -

15-213, F'07

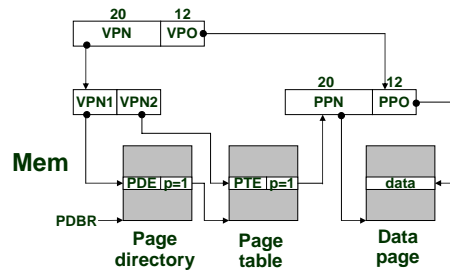
P6 Page Table Translation



- 14 -

15-213, F'07

Translating with the P6 Page Tables (case 1/1)



Case 1/1: page table and page present.

MMU Action:

- MMU builds physical address and fetches data word.

OS action

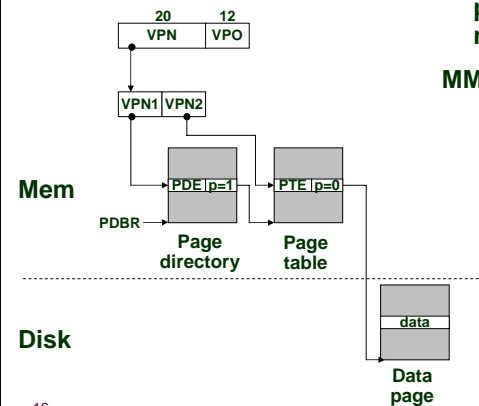
- None

Disk

- 15 -

15-213, F'07

Translating with the P6 Page Tables (case 1/0)



Case 1/0: page table present but page missing.

MMU Action:

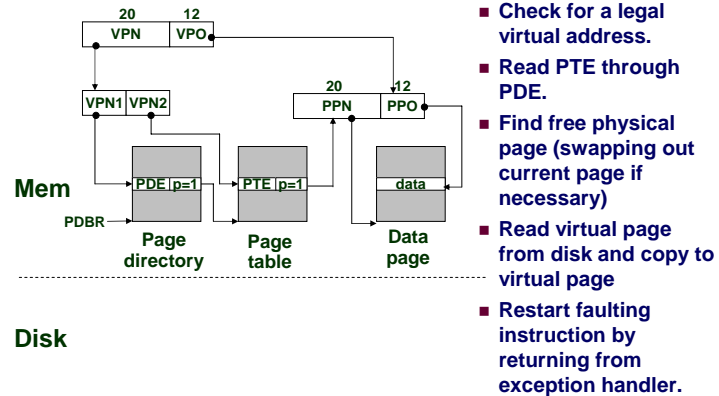
- Page fault exception
- Handler receives the following args:
 - VA that caused fault
 - Fault caused by non-present page or page-level protection violation
 - Read/write
 - User/supervisor

Disk

- 16 -

15-213, F'07

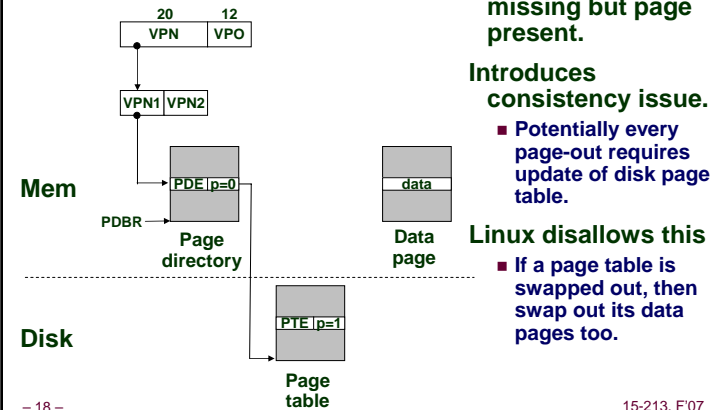
Translating with the P6 Page Tables (case 1/0, cont)



- 17 -

15-213, F'07

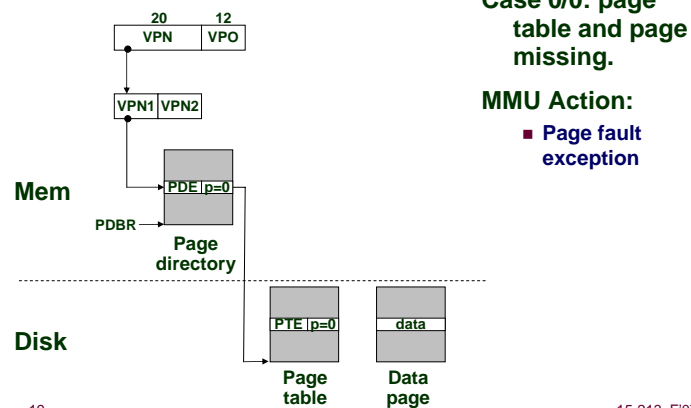
Translating with the P6 Page Tables (case 0/1)



- 18 -

15-213, F'07

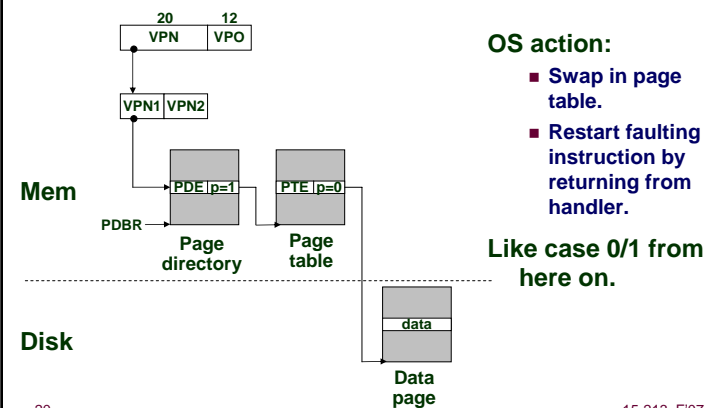
Translating with the P6 Page Tables (case 0/0)



- 19 -

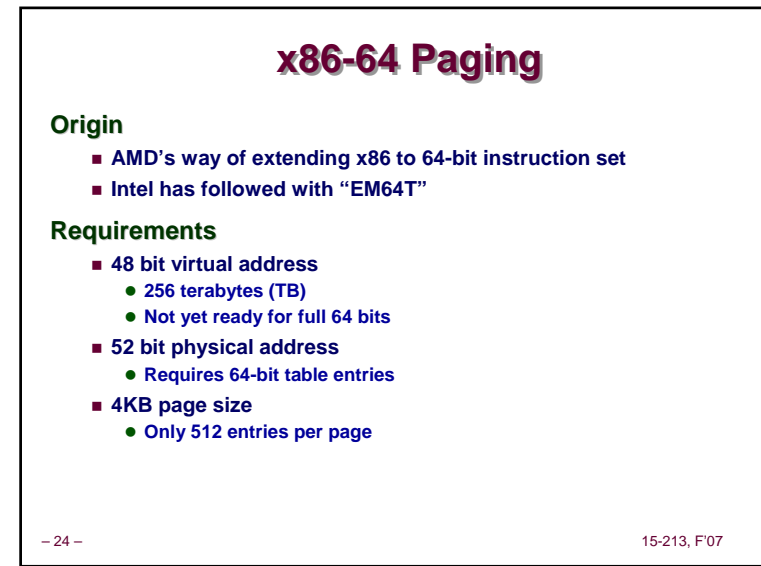
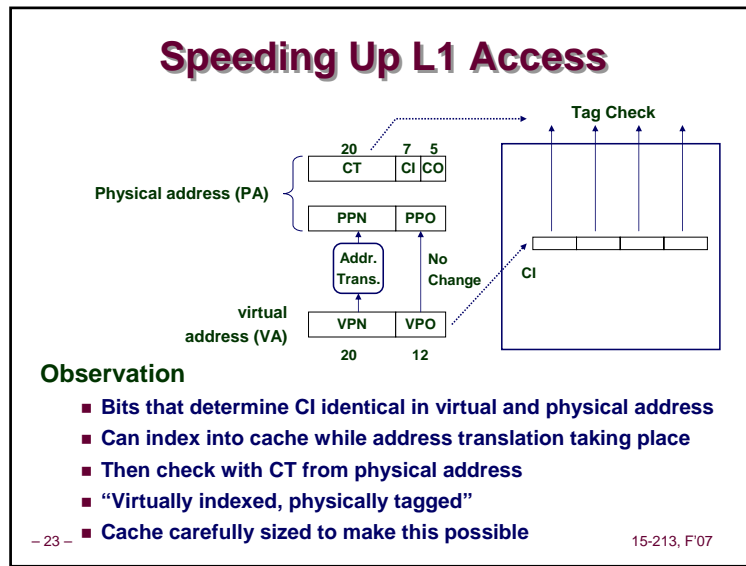
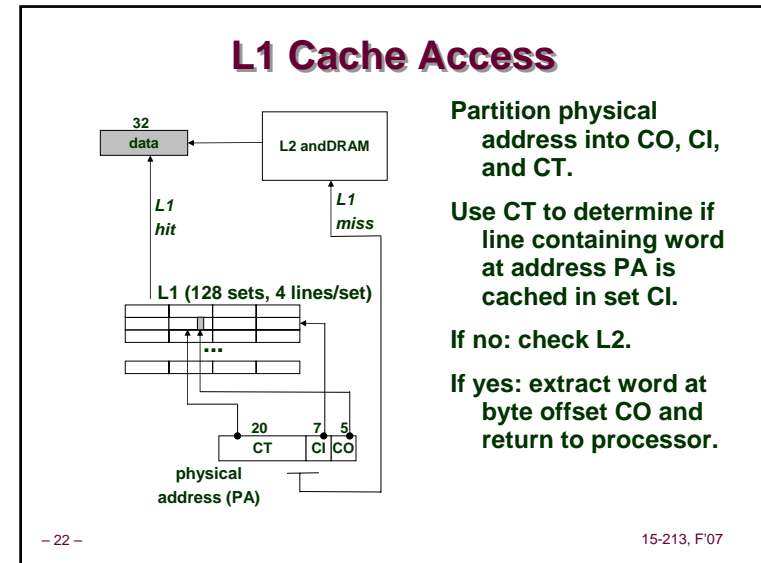
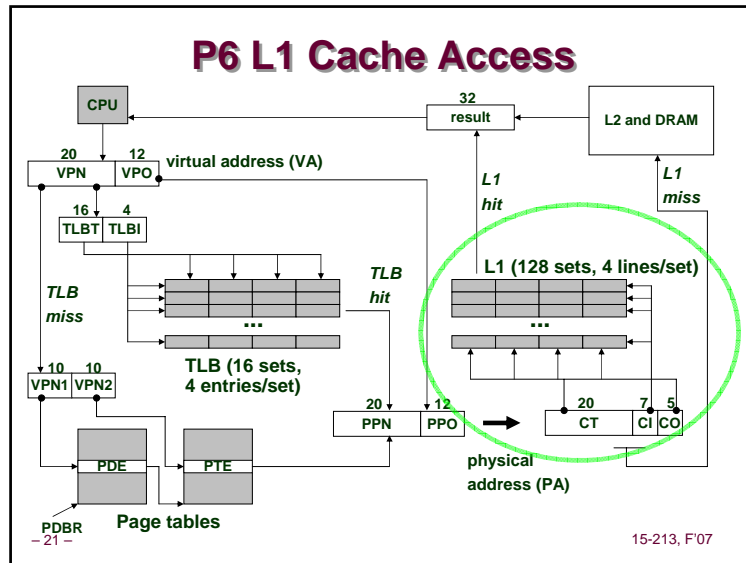
15-213, F'07

Translating with the P6 Page Tables (case 0/0, cont)

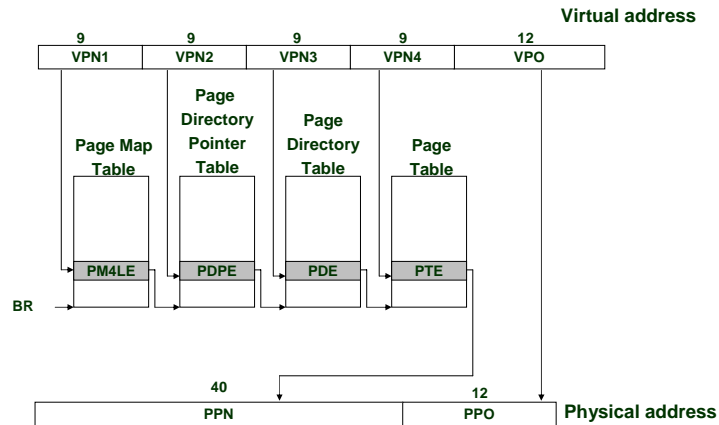


- 20 -

15-213, F'07



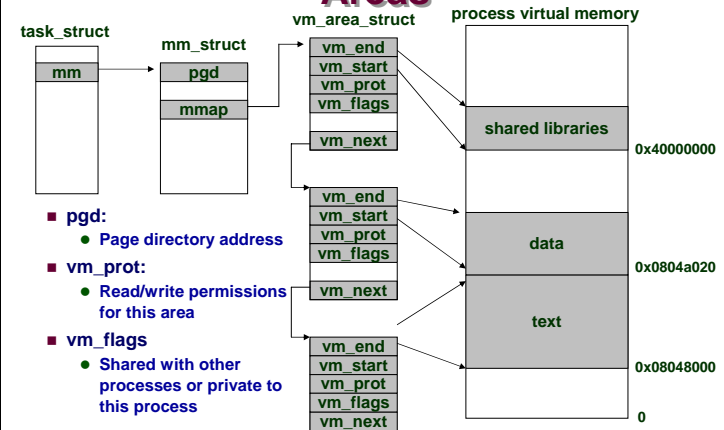
x86-64 Paging



- 25 -

15-213, F'07

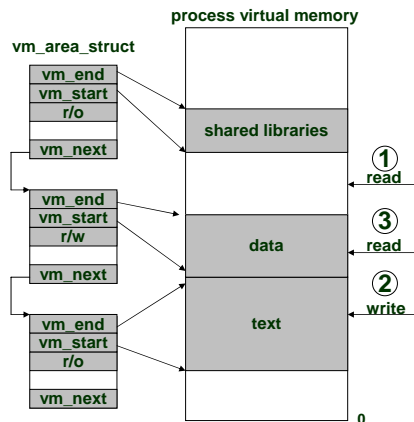
Linux Organizes VM as Collection of "Areas"



- 26 -

15-213, F'07

Linux Page Fault Handling



- 27 -

15-213, F'07

Is the VA legal?

- i.e., Is it in an area defined by a vm_area_struct?
- If not then signal segmentation violation (e.g. (1))

Is the operation legal?

- i.e., Can the process read/write this area?
- If not then signal protection violation (e.g., (2))

If OK, handle fault

- e.g., (3)

Memory Mapping

Creation of new VM area done via "memory mapping"

- Create new vm_area_struct and page tables for area
- Area can be backed by (i.e., get its initial values from) :
 - Regular file on disk (e.g., an executable object file)
 - » Initial page bytes come from a section of a file
 - Nothing (e.g., bss)
 - » Initial page bytes are zeros
- Dirty pages are swapped back and forth between a special swap file.

Key point: no virtual pages are copied into physical memory until they are referenced!

- Known as "demand paging"
- Crucial for time and space efficiency

- 28 -

15-213, F'07

User-Level Memory Mapping

```
void *mmap(void *start, int len,
           int prot, int flags, int fd, int offset)
```

- Map `len` bytes starting at offset `offset` of the file specified by file description `fd`, preferably at address `start` (usually 0 for don't care).
 - `prot`: MAP_READ, MAP_WRITE
 - `flags`: MAP_PRIVATE, MAP_SHARED
- Return a pointer to the mapped area.
- Example: fast file copy
 - Useful for applications like Web servers that need to quickly copy files.
 - `mmap` allows file transfers without copying into user space.

- 29 -

15-213, F'07

mmap() Example: Fast File Copy

```
#include <unistd.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int main() {
    struct stat stat;
    int i, fd, size;
    char *bufp;

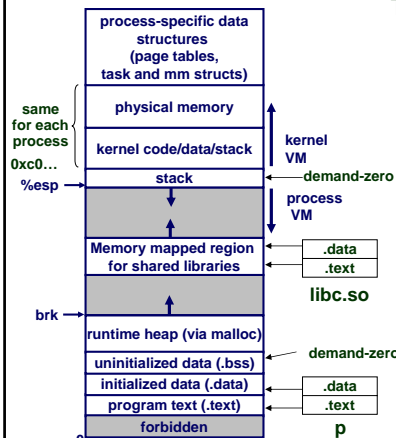
    /* open the file & get its size*/
    fd = open("./mmap.c", O_RDONLY);
    fstat(fd, &stat);
    size = stat.st_size;
    /* map the file to a new VM area */
    bufp = mmap(0, size, PROT_READ,
               MAP_PRIVATE, fd, 0);

    /* write the VM area to stdout */
    write(1, bufp, size);
}
```

- 30 -

15-213, F'07

Exec() Revisited



- 31 -

15-213, F'07

To run a new program `p` in the current process using `exec()`:

- Free `vm_area_struct`'s and page tables for old areas.
- Create new `vm_area_struct`'s and page tables for new areas.
 - Stack, `bss`, `data`, `text`, shared libs.
 - Text and data backed by ELF executable object file.
 - `bss` and stack initialized to zero.
- Set PC to entry point in `.text`
 - Linux will swap in code and data pages as needed.

Fork() Revisited

To create a new process using `fork()`:

- Make copies of the old process's `mm_struct`, `vm_area_struct`'s, and page tables.
 - At this point the two processes are sharing all of their pages.
 - How to get separate spaces without copying all the virtual pages from one space to another?
 - » "copy on write" technique.
- Copy-on-write
 - Make pages of writeable areas read-only
 - Flag `vm_area_struct`'s for these areas as private "copy-on-write".
 - Writes by either process to these pages will cause page faults.
 - » Fault handler recognizes copy-on-write, makes a copy of the page, and restores write permissions.

Net result:

- Copies are deferred until absolutely necessary (i.e., when one of the processes tries to modify a shared page).

- 32 -

15-213, F'07

Memory System Summary

Cache Memory

- Purely a speed-up technique
- Behavior invisible to application programmer and (mostly) OS
- Implemented totally in hardware

Virtual Memory

- Supports many OS-related functions
 - Process creation
 - Task switching
 - Protection
- Combination of hardware & software implementation
 - Software management of tables, allocations
 - Hardware access of tables
 - Hardware caching of table entries (TLB)