

15-213

P6/Linux Memory System March 26, 2002

Topics

- P6 address translation
- Linux memory management
- Linux page fault handling
- memory mapping
- Reading: 10.7-10.8
- Problems: 10.14

class20.ppt

Intel P6

Internal Designation for Successor to Pentium

- Which had internal designation P5

Fundamentally Different from Pentium

- Out-of-order, superscalar operation
- Designed to handle server applications
 - Requires high performance memory system

Resulting Processors

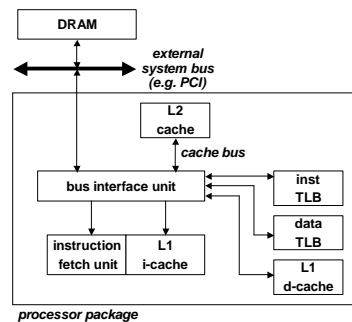
- PentiumPro (1996)
- Pentium II (1997)
 - Incorporated MMX instructions
 - » special instructions for parallel processing
 - L2 cache on same chip
- Pentium III (1999)
 - Incorporated Streaming SIMD Extensions
 - » More instructions for parallel processing

class20.ppt

– 2 –

15-213 S'02 (Based on CS 213 F'01)

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

class20.ppt

– 3 –

15-213 S'02 (Based on CS 213 F'01)

Review of abbreviations

Symbols:

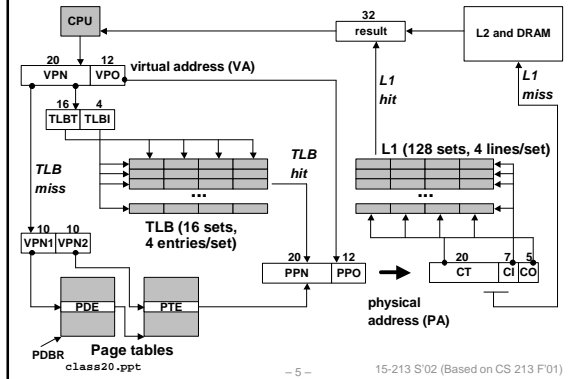
- Components of the virtual address (VA)
 - TLBI: TLB index
 - TLBT: TLB tag
 - VPO: virtual page offset
 - VPN: virtual page number
- Components of the physical address (PA)
 - PPO: physical page offset (same as VPO)
 - PPN: physical page number
 - CO: byte offset within cache line
 - CI: cache index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

class20.ppt

– 4 –

15-213 S'02 (Based on CS 213 F'01)

Overview of P6 address translation



- 5 -

15-213 S'02 (Based on CS 213 F'01)

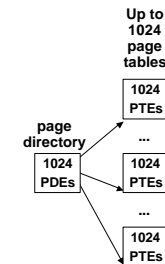
P6 2-level page table structure

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.



class20.ppt

- 6 -

15-213 S'02 (Based on CS 213 F'01)

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

class20.ppt

- 7 -

15-213 S'02 (Based on CS 213 F'01)

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)

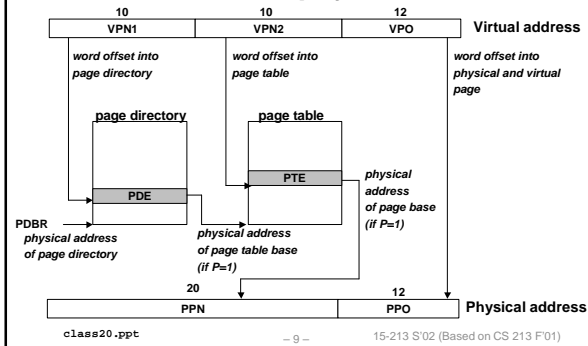
31	1	0
Available for OS (page location in secondary storage)	P=0	

class20.ppt

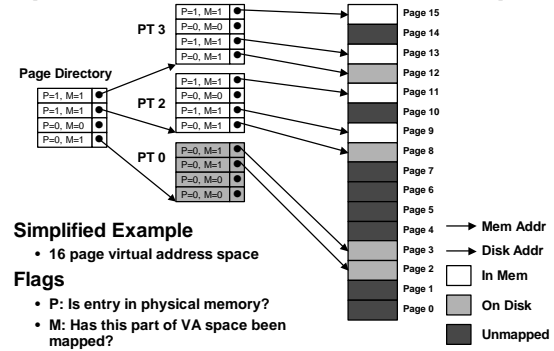
- 8 -

15-213 S'02 (Based on CS 213 F'01)

How P6 page tables map virtual addresses to physical ones



Representation of Virtual Address Space



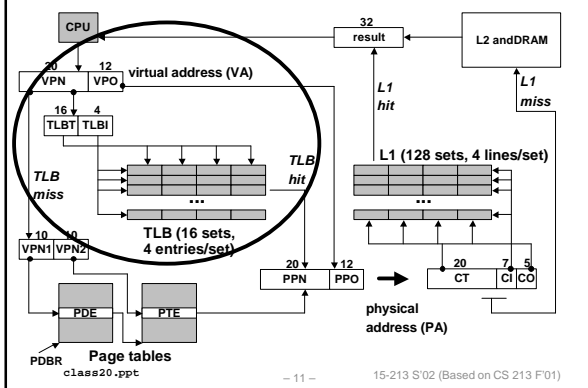
Simplified Example

- 16 page virtual address space

Flags

- P: Is entry in physical memory?
- M: Has this part of VA space been mapped?

P6 TLB translation



P6 TLB

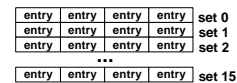
TLB entry (not all documented, so this is speculative):



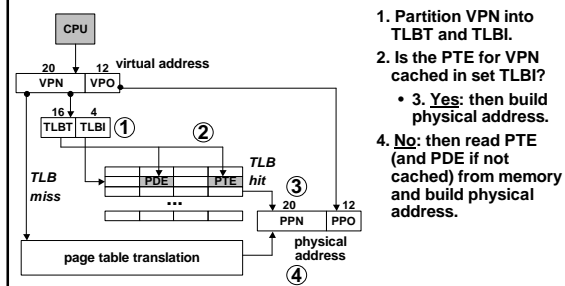
- V: indicates a valid (1) or invalid (0) TLB entry
- PD: is this entry a PDE (1) or a PTE (0)?
- tag: disambiguates entries cached in the same set
- PDE/PTE: page directory or page table entry

Structure of the data TLB:

- 16 sets, 4 entries/set



Translating with the P6 TLB



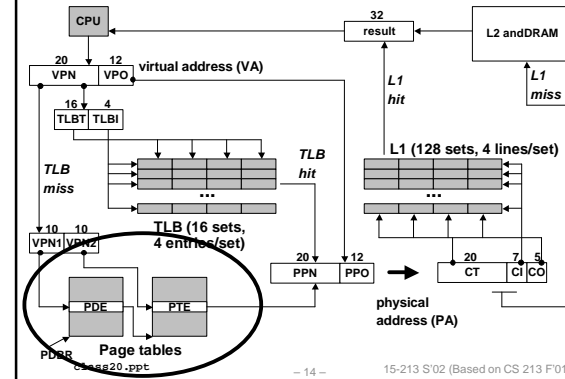
1. Partition VPN into TLBT and TLBI.
2. Is the PTE for VPN cached in set TLBI?
 - 3. Yes: then build physical address.
4. No: then read PTE (and PDE if not cached) from memory and build physical address.

class20.ppt

- 13 -

15-213 S'02 (Based on CS 213 F'01)

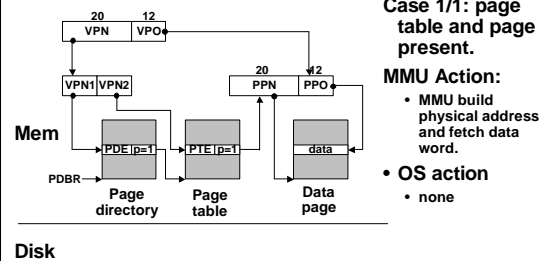
P6 page table translation



- 14 -

15-213 S'02 (Based on CS 213 F'01)

Translating with the P6 page tables (case 1/1)



Case 1/1: page table and page present.

MMU Action:

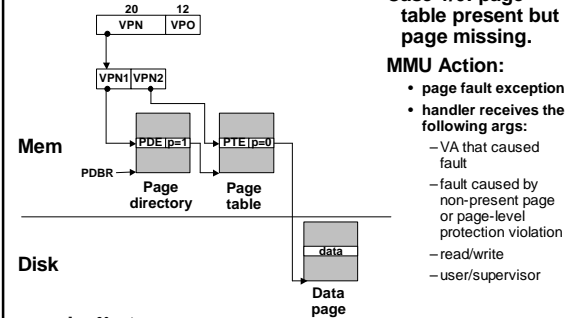
- MMU build physical address and fetch data word.
- OS action
- none

class20.ppt

- 15 -

15-213 S'02 (Based on CS 213 F'01)

Translating with the P6 page tables (case 1/0)



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

MMU Action:

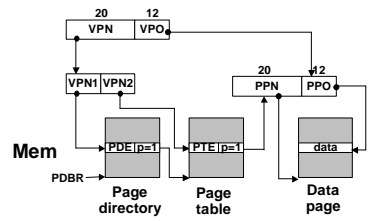
- page fault exception
- handler receives the following args:
 - VA that caused fault
 - fault caused by non-present page or page-level protection violation
 - read/write
 - user/supervisor

class20.ppt

- 16 -

15-213 S'02 (Based on CS 213 F'01)

Translating with the P6 page tables (case 1/0, cont)



Disk

class20.ppt

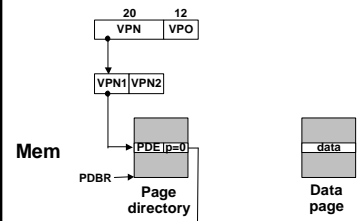
- 17 -

15-213 S'02 (Based on CS 213 F'01)

OS Action:

- Check for a legal virtual address.
- Read PTE through PDE.
- Find free physical page (swapping out current page if necessary)
- Read virtual page from disk and copy to virtual page
- Restart faulting instruction by returning from exception handler.

Translating with the P6 page tables (case 0/1)



Disk

class20.ppt

- 18 -

15-213 S'02 (Based on CS 213 F'01)

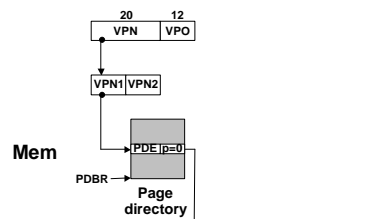
Case 0/1: page table missing but page present.
Introduces consistency issue.

- potentially every page out requires update of disk page table.

Linux disallows this

- if a page table is swapped out, then swap out its data pages too.

Translating with the P6 page tables (case 0/0)



Disk

class20.ppt

- 19 -

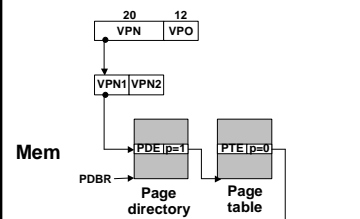
15-213 S'02 (Based on CS 213 F'01)

Case 0/0: page table and page missing.

MMU Action:

- page fault exception

Translating with the P6 page tables (case 0/0, cont)



Disk

class20.ppt

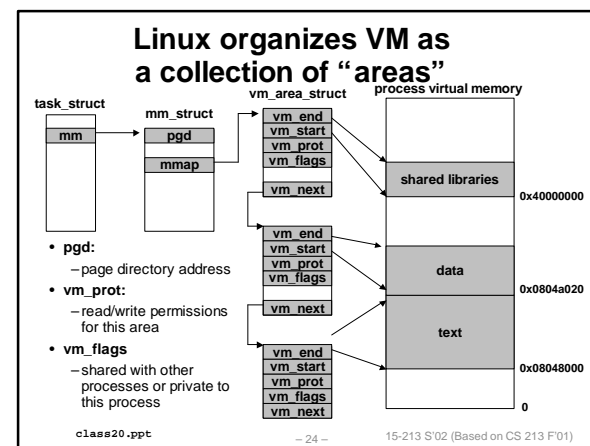
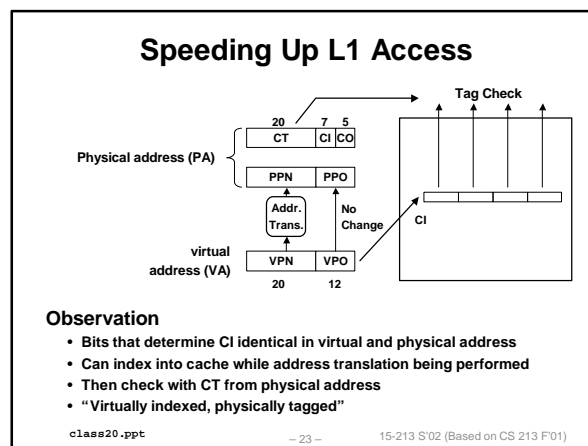
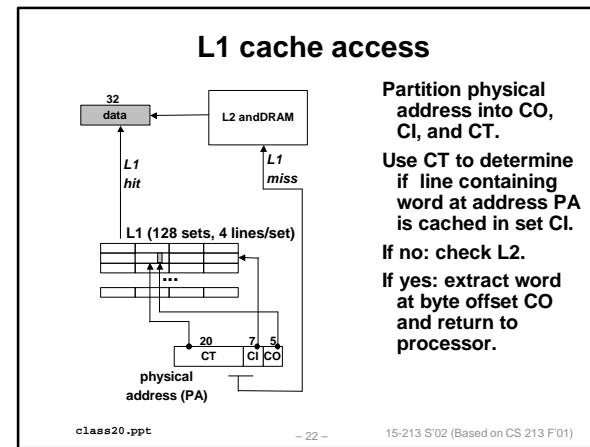
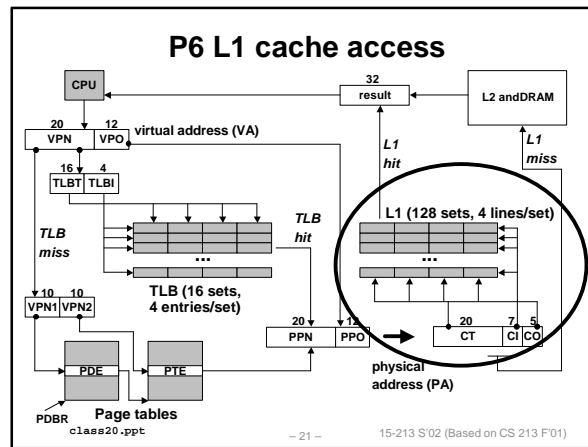
- 20 -

15-213 S'02 (Based on CS 213 F'01)

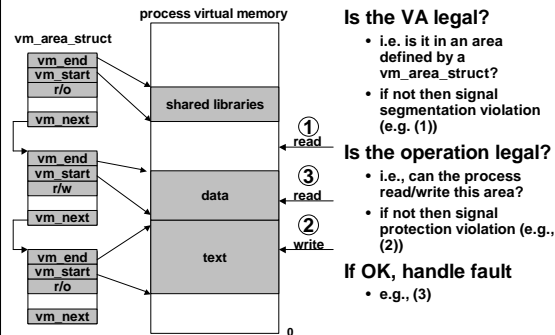
OS action:

- swap in page table.
- restart faulting instruction by returning from handler.

Like case 0/1 from here on.



Linux page fault handling



class20.ppt

- 25 -

15-213 S'02 (Based on CS 213 F'01)

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

class20.ppt

- 26 -

15-213 S'02 (Based on CS 213 F'01)

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.

class20.ppt

- 27 -

15-213 S'02 (Based on CS 213 F'01)

mmap() example: fast file copy

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

/*
 * mmap.c - a program that uses mmap
 * to copy itself to stdout
 */
int main() {
    struct stat stat;
    int i, fd, size;
    char *bufp;

    /* open the file and 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);

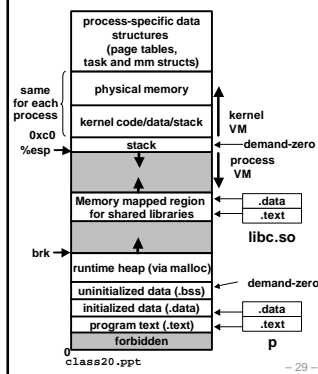
    /* write the VM area to stdout */
    write(1, bufp, size);
}
```

class20.ppt

- 28 -

15-213 S'02 (Based on CS 213 F'01)

Exec() revisited



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.

class20.ppt

– 29 –

15-213 S'02 (Based on CS 213 F'01)

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).

class20.ppt

– 30 –

15-213 S'02 (Based on CS 213 F'01)

Memory System Summary

Cache Memory

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

Virtual Memory

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

class20.ppt

– 31 –

15-213 S'02 (Based on CS 213 F'01)