CS 61C Spring 2024

Virtual Memory

Discussion 12

1 Precheck

This section is designed as a conceptual check for you to determine if you conceptually understand and have any misconceptions about this topic. Please answer true/false to the following questions, and include an explanation:

1.1 Having virtual memory helps protect a system.

True. By dedicating specific pages to a program, the OS can ensure that a program does not access pages it's not been given access to, providing isolation between programs.

1.2 The virtual address space is limited by the amount of memory in the system.

False. The physical address space is limited by the amount of physical memory in the system, the size of the virtual address space is set by the OS.

1.3 The virtual and physical page number must be the same size.

False. There could be fewer physical pages than virtual pages. However, the page size does need to be the same.

1.4 If a page table entry can not be found in the TLB, then a page fault has occurred.

False, the TLB acts as a cache for the page table, so an item can be valid in page table but not stored in TLB. A page fault occurs either when a page cannot be found in the page table or it has an invalid bit.

2 Addressing

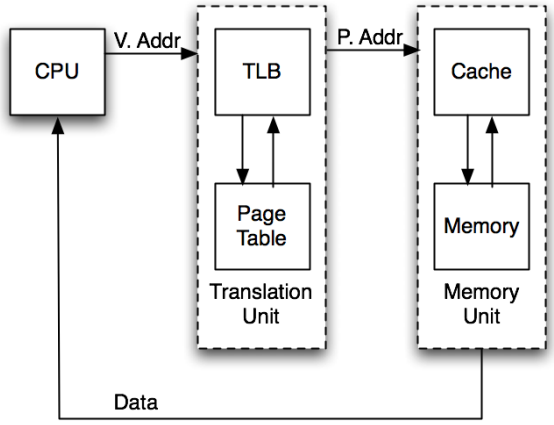
Virtual Address (VA) What your program uses

Virtual Page Number (VPN)	Page Offset

Physical Address (PA) What actually determines where in memory to go

Physical Page Number (PPN)	Page Offset

For example, with 4 KiB pages and byte addresses, there are 12 page offset bits since 4 KiB = 2^{12} B = 4096 B.



Pages

A chunk of memory or disk with a set size. Addresses in the same virtual page map to addresses in the same physical page. The page table determines the mapping.

Valid	Dirty	Permission Bits	PPN
— Page entry (VPN: 0) —			
— Page entry (VPN: 1) —			

Each stored row of the page table is called a **page table entry**. There are $2^{\text{VPN bits}}$ such entries in a page table. Say you have a VPN of 5 and you want to use the page table to find what physical page it maps to; you'll check the 5th (0-indexed) page table entry. If the valid bit is 1, then that means that the entry is valid (in other words, the physical page corresponding to that virtual page is in main memory as opposed to being only on disk) and therefore you can get the PPN from the entry and access that physical page in main memory.

The page table is stored in memory: the OS sets a register (the Page Table Base Register) telling the hardware the address of the first entry of the page table. If you write to a page in memory, the processor updates the "dirty" bit in the page table entry corresponding to that page, which lets the OS know that updating that page on disk is necessary (remember: main memory contains a subset of what's on disk). This is a similar concept as having a dirty bit for each cache block in a write-back cache. Each process gets its own illusion of full memory to work with, and therefore its own page table.

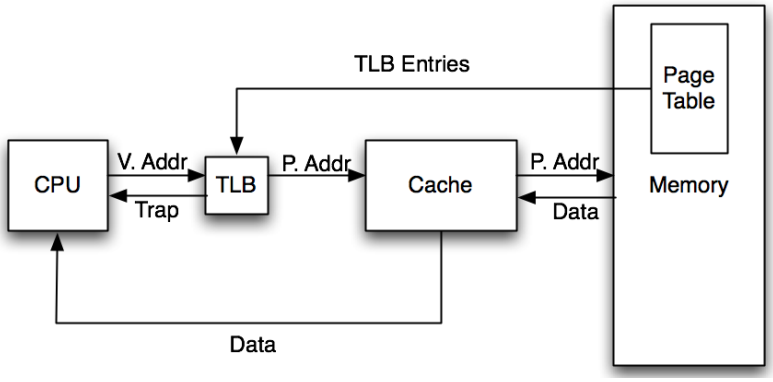
Protection Fault The page table entry for a virtual page has permission bits that prohibit the requested operation. This is how a segmentation fault occurs.

Page Fault The page table entry for a virtual page has its valid bit set to false. This means that the entry is not in memory. For simplicity, we will assume the address causing the page fault is a valid request, and maps to a page that was swapped from memory to disk. Since the requested address is valid, the operating system checks if the page exists on disk. If so, we transfer the page to memory (evicting another page if necessary), and add the mapping to the page table **and** the translation lookaside buffer (TLB).

Translation Lookaside Buffer

A cache for the page table. Each block is a single page table entry. If an entry is not in the TLB or the valid bit = 0, it's a TLB miss. Typically fully associative:

TLB Valid	Tag (VPN)	Page Table Entry		
		Page Dirty	Permission Bits	PPN
— *TLB entry* —				
— *TLB entry* —				



To access some memory location, we get the virtual page number (VPN) from the virtual address (VA) and first try to translate the VPN to a physical page number (PPN) using the translation lookaside buffer (TLB). If there is a TLB miss, we check the page table for the VPN to PPN mapping. (remember: the TLB is a subset of the page table!). If the valid bit = 0, then this is a page fault; memory doesn't contain the corresponding physical page! This means we need to fetch the physical page from disk and put it into memory, update the page table entry, and load the entry into the TLB, Then, we use the physical page and the offset of the physical address in the page to access memory as the program intended.

2.1 What are three specific benefits of using virtual memory?

- Illusion of access to entire address space (bridges memory and disk in memory hierarchy).
- Avoids memory address conflict between programs by simulating a separate full address space for each process, so that the linker/loader don't need to know about other programs.
- Enforces protection between processes and even within a process (e.g. read-only pages set up by the OS).

2.2 What should happen to the TLB when a new value is loaded into the page table address register (i.e. we are switching page tables to those for another process)?

The valid bits of the TLB should all be set to 0. The page table entries in the TLB corresponded to the old process/page table, so none of them are valid once the page table address register points to a different page table.

3 VM Access Patterns

3.1 A processor has 16-bit addresses, 256 byte pages, and an 8-entry fully associative TLB with LRU replacement (the LRU field is 3 bits and encodes the order in which pages were accessed, 0 being the most recent). At some time instant, the TLB for the current process is the initial state given in the table below, and we have three free physical pages as given below. Assume that all current page table entries are in the initial TLB. Assume also that all pages can be read from and written to. Fill in the final state of the TLB according to the following access pattern, and also write out the physical addresses corresponding to each location accessed.

Free Physical Pages `0x17`, `0x18`, `0x19`

Access Pattern

1. `0x11f0` (**Read**)
2. `0x1301` (**Write**)
3. `0x20ae` (**Write**)
4. `0x2332` (**Write**)
5. `0x20ff` (**Read**)
6. `0x3415` (**Write**)

Initial TLB

VPN	PPN	Valid	Dirty	LRU
`0x01`	`0x11`	1	1	0
`0x00`	`0x00`	0	0	7
`0x10`	`0x13`	1	1	1
`0x20`	`0x12`	1	0	5
`0x00`	`0x00`	0	0	7
`0x11`	`0x14`	1	0	4
`0xac`	`0x15`	1	1	2
`0xff`	`0xff`	1	0	3

Final TLB

VPN	PPN	Valid	Dirty
`0x01`	`0x11`	1	1
`0x13`	`0x17`	1	1
`0x10`	`0x13`	1	1
`0x20`	`0x12`	1	1
`0x23`	`0x18`	1	1
`0x11`	`0x14`	1	0
`0xac`	`0x15`	1	1
`0x34`	`0x19`	1	1

1. `0x11f0` (**Read**): hit, PA: 0x14f0; LRUs: 1, 7, 2, 5, 7, 0, 3, 4

2. `0x1301` (**Write**): miss, map VPN `0x13` to PPN `0x17`, set valid and dirty, PA: 0x1701; LRUs: 2, 0, 3, 6, 7, 1, 4, 5
3. `0x20ae` (**Write**): hit, set dirty, PA: 0x12ae; LRUs: 3, 1, 4, 0, 7, 2, 5, 6
4. `0x2332` (**Write**): miss, map VPN `0x23` to PPN `0x18`, set valid and dirty, PA: 0x1832; LRUs: 4, 2, 5, 1, 0, 3, 6, 7
5. `0x20ff` (**Read**): hit, PA: 0x12ff; LRUs: 4, 2, 5, 0, 1, 3, 6, 7
6. `0x3415` (**Write**): miss and replace last entry, map VPN `0x34` to `0x19`, set dirty, PA: 0x1915; LRUs, 5, 3, 6, 1, 2, 4, 7, 0