CS61C Spring 2026

Precheck: Virtual Memory

Discussion 12

1 Pre-Check: T/F?

1.1 Having virtual memory helps protect a system.

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

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

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

2 Virtual Memory

Virtual Memory defines two types of *address spaces* (set of addresses for all available memory locations):

(a) Virtual Address Space: the set of addresses for our program
(b) Physical Address Space: the set of addresses that map to physical locations in memory

An analogy is that a PDF's page 6 is a photocopy of page 200 of a physical book. We would say that page 6 is the *virtual address* that maps to the corresponding *physical address* of 200.

Address Translation

Addresses are converted between their virtual and physical mapping by a **page table** unique to each process.

Virtual Address (VA): An X-bit address which is what your program uses. It comprises of:

Virtual Page Number (VPN)	Page Offset

Physical Address (PA): A Y-bit address which is the physical location of memory. It comprises of:

Physical Page Number (PPN)	Page Offset

Page tables are a lookup table (managed by the OS) for the VPN → PPN mapping. Page Offset is similar to block offset for caches, and must be the same size in both Virtual and Physical addresses. However, the number of VPN and PPN bits may differ from each other (think about why that's the case)!

Memory Pages

Modern architectures segment our memory and disk into chunks of a set size called **pages**. 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.

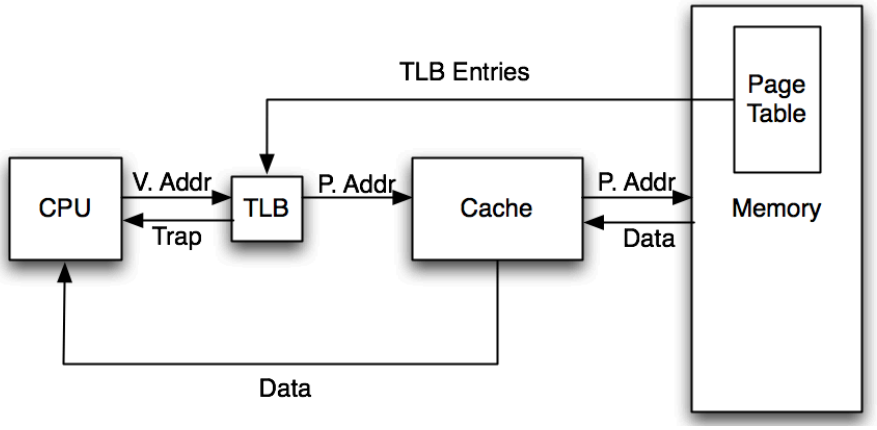
Page Fault

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 (TLB)

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:

<table>
<tr><td rowspan="2">TLB Valid</td><td rowspan="2">Tag (VPN)</td><td colspan="3">Page Table Entry</td></tr>
<tr><td>Page Dirty</td><td>Permission Bits</td><td>PPN</td></tr>
<tr><td colspan="5">— TLB entry —</td></tr>
<tr><td colspan="5">— TLB entry —</td></tr>
</table>



To access some memory location, we do the following:

(a) Get the virtual page number (VPN) from the virtual address (VA) and try translate the VPN to a physical page number (PPN) using the translation lookaside buffer (TLB)
(b) 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!).

(c) 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.

(d) Then, we use the physical page and the offset of the physical address in the page to access memory as the program intended.