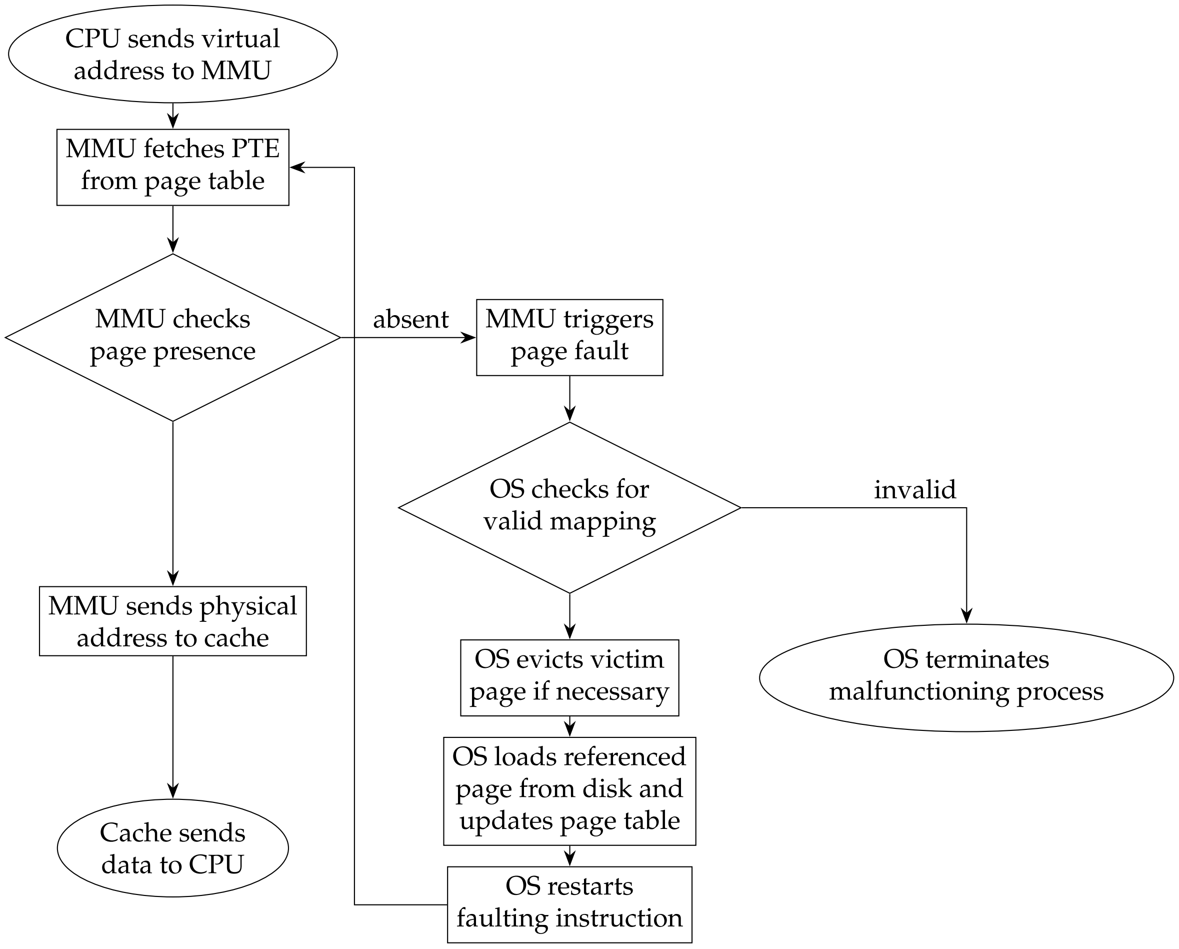
Introduction

This activity will help you understand the details of address translation and memory mapping in a virtual memory system.

1. Address Translation Concepts

Problems: 7 / Allocated Time: 10 minutes

This flowchart illustrates the steps required to read a value from memory on a system with virtual memory.


CPU sends virtual address to MMU
MMU fetches PTE from page table
MMU checks page presence
absent
MMU triggers page fault
OS checks for valid mapping
invalid
OS terminates malfunctioning process
MMU sends physical address to cache
OS evicts victim page if necessary
OS loads referenced page from disk and updates page table
OS restarts faulting instruction
Cache sends data to CPU

The problems in this section discuss the details of these steps.

Problem 1. For a simple system with a one-level page table, what sub-steps does the MMU take when it fetches a PTE from a page table? (Hint: think about how to split up the address in order to look for an entry in the page table.)

Given $N = 2^n$ addresses in the virtual address space, $M = 2^m$ addresses in the physical address space, and $P = 2^p$ the page size in bytes, the VPN is defined by the leftmost $n - p$ bits of the address. In a system with a one-level page table, the VPN can be used directly as an array index in the page table.

Problem 2. The MMU must know the *physical* address of the page table in order to read page table entries from memory. Why does it need a physical address?

If the MMU knew only a *virtual* address for the page table then, in order to read from the page table, it would first need to look up the physical address of the page table, in the page table, . . .

Problem 3. Why are one-level page tables impractical? How do multi-level page tables fix this problem?

In typical systems, a single-level page table covering the entire address space would be impractically large. For instance, with a page size of 4KB (2^{12} bytes), a 48-bit address space, and an 8-byte PTE, a single-level page table would occupy 512 GB, which is more memory than most computers have.

A multi-level page table takes advantage of the fact that, in a typical virtual address space, most of the addresses are not mapped to physical addresses (we say the mapping is *sparse*.) A multi-level table can represent large regions of unmapped addresses with a single "no mapping" entry in an upper level of the table, rather than having individual "no mapping" PTEs for every page in the region.

Problem 4. Why might an MMU take longer to look up addresses that *are* mapped in a multi-level page table than the equivalent one-level page table?

A k-level page table requires k memory loads, in order to determine the physical address. There is no spatial locality to these loads.

Problem 5. What is the Translation Lookaside Buffer (TLB) and what problem is it intended to solve? At what point in the process of fetching PTEs is the TLB used?

The TLB is a small set-associative cache dedicated to storing mappings from virtual to physical addresses. It mitigates the cost of lookups in a multi-level page table. The MMU consults the TLB for each address as its first action; if there is a TLB hit, it does not need to fetch anything from the page table.

Problem 6. When does a page fault happen? What does the page fault handler do when it is called?

The MMU triggers a page fault whenever the CPU tries to access a virtual address that is marked "not present" in the page table. The page fault handler will determine *why* the page is not present. If it never existed at all, the process is malfunctioning and will be terminated. But if the page has been swapped out or has not yet been loaded or created, then the handler will find a physical page to assign to that virtual address (evicting other data if necessary), fill it with the appropriate data, update the page tables, and resume execution.

Problem 7. How does virtual memory interact with the memory cache(s)?

The purpose of memory caching is to speed up access to *whatever* data is most frequently used. In a block diagram of the computer, the MMU sits "in between" the CPU and the memory cache(s), which work only with physical addresses. This means that data from multiple processes may coexist in the cache (or compete for cache space).

2. TLB Practice

Questions: 1 / Allocated Time: 7 minutes

Problem 8. Assume a system with a two-way set-associative TLB, with a total of eight entries. Virtual and physical addresses are both 16 bits, and pages are 2^8 bytes each.

At some point in a program's execution, the contents of the TLB are as follows:

Index	Tag	PPN	Valid
0	`0x13`	`0x30`	1
0	`0x34`	`0x58`	1
1	`0x1F`	`0x80`	0
1	`0x2A`	`0x72`	1
2	`0x1F`	`0x95`	1
2	`0x20`	`0xAA`	0
3	`0x3F`	`0x20`	1
3	`0x3E`	`0xFF`	0

Based on the contents of the TLB, fill in the following table of virtual to physical address mappings. If you don't have enough information to fill a cell, write "?" in that cell.

Virtual address	Physical address
0x7E85	0x9585
0xD301	?
0x4C20	0x3020
0xD040	?
?	0x5830

See `http://www.cs.cmu.edu/afs/cs/academic/class/15213-m19/www/activities/213_lecture18-sol.pdf` for additional guidance.

3. Memory Mapping

Questions: 2 / Allocated Time: 7 minutes

Problem 9. Describe two different things that can provide the "backing store" for a region of virtual memory, and explain what the page fault handler does the *first time* a page of memory with those types of backing store is accessed.

The most common two types of backing store are files on disk and the swap space (which is also on disk, but not associated with any named file; it's the equivalent of scratch paper).

Whenever a page of memory is first accessed, the page fault handler fills that page with initial data. For pages backed by a file, the initial data comes from the file's contents. For pages backed by swap space, the initial data is all bytes zero.

Problem 10. Each process has its own, private virtual address space. However, some parts of the address space might be *shared* with other processes—that is, mapped to the same physical addresses in all the processes that share that region. Give an example of when this happens and explain why it might improve overall system performance.

One common situation where a region of the address space is shared among processes is when several processes all map a region to the same file. For instance, they might all use the same dynamic library. This may improve system performance because there only needs to be one copy of the file in memory no matter how many processes are referring to it.

A. Appendix: Address Translation Symbol Reference

- Basic parameters
  - $N = 2^n$: Number of addresses in virtual address space
  - $M = 2^m$: Number of addresses in physical address space
  - $P = 2^p$: Page size (bytes)
- Components of the virtual address (VA)
  - VPO: Virtual page offset
  - VPN: Virtual page number
  - TLBI: TLB index
  - TLBT: TLB tag
- 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