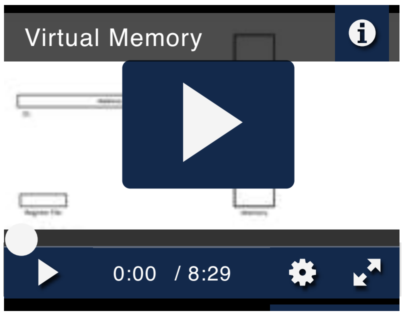
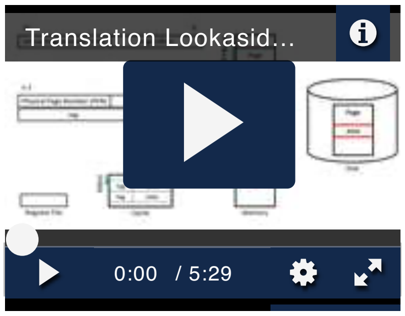
PRE24.1. 24 Text and Videos

Virtual Memory Videos





Disks (will see only on this pre-flight): [Disks Slides]

Up until now, we have made two simplifying assumptions about the ways processors work: 1) each program we write will be the only program running on a processor and 2) the code and the data each program uses just magically appears and is small enough to fit in main memory. The first assumption is problematic because we do want multiple programs running at the same time. These different programs may want to write to the same memory addresses, meaning that diffrent programs could theoretically overwrite each other's data in main memory. The second assumption is problematic because even though our main memory is large, it is not large enough to contain all data/instructions we could possibly want to use AND main memory is volatile, meaning that all data is lost when the memory loses power (ideally we don't want to lose all programs and data from our computers when we turn them off!). Consequently, we need a system that can flexibly 1) allow multiple programs to access data memory while not allowing them to access each other's data /instrucitons (i.e., provide data security) and 2) allow our assembly code programs to pretend that data is in main memory but is actually copying data from even larger, non-volatile sources (like hard disks) to main memory without our program knowing. Enter Virtual Memory.

PRE 24

Assessment overview

Total points:	0/50
Score:	0%

Question PRE24.1

Value:	1
Total points:	— /1
Auto-graded questior	

Previous question

Next question

Personal Notes

No attached notes

Attach a file

Add text note

Virtual Memory solves both problems by using the principle of indirection, treating the addresses that our programs think they are writing to as references to the real addresses where their data is stored. When using a load or store instruction, the address you construct is actually a virtual address. The virtual memory system converts this virtual address into a physical address, which is actually where your data is stored, by looking up the virtual address in a special lookup table called the page table that tells it the correct physical address for each virtual address for each process (program) that is running. For example, if your program tries to read from address 0x10010008 (i.e., `lw $t0, 0x10010008`), that is the virtual address. The virtual memory system would look up 0x10010008 in the page table and might find that the physical address is actually 0xcafe0008. Likewise, a different program might try to read from address 0x10010008 for its virtual address but the page table would tell the virtual memory system that the physical address for that program is actually 0x02330008. Although both programs think they are reading from the same address, virtual memory lets both programs read their own data separate from each other.

If the virtual memory system does not find the desired virtual address in the page table, it causes a page fault (similar to a cache miss). A page fault means that the desired data has not yet been copied into the memory. After a page fault, the desired data is located on the disk, copied to the memory, and a new entry is added to the page table for future accesses to that entry in the page table. The virtual memory system can then properly convert the virtual address into a physical address so that programs can securely access their data.

The videos and other resources will dive a little deeper into how we actually implement the virtual memory system.

Cache Coherence Videos

First, a few important definitions

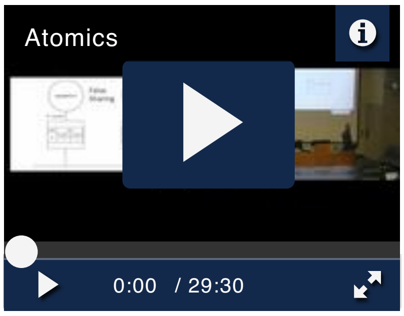
- Processor Core: A datapath like what we have seen during class that has an instruction cache, register file, alu, data cache, etc.
- Thread: A program or a small part of a program that can be run independently and scheduled indepedently by the operating system
- Cache Coherence: The caches from the different cores are said to have coherence for a piece of data when they all agree on the current value of that piece of data or do not have a value stored for that data.

In a single core processor, the operating system creates the apperance of running multiple programs at a time by quickly switching between different threads based on what processes are in demand by the user. By adding multiple cores in a processor, we create the ability to acutally run multiple threads in parallel with each core running threads from different processes or several threads from the same process. This parallelism can dramatically improve performance but at the cost of some new concerns about how to manage dependencies.

A critical challenge in effectively writing a single program that can concurrently run multiple threads is that each core has its own cache. If we perform a write to the cache in one core, the caches of the other cores will not also be written to, causing the caches to lose coherence. Therefore, we need protocols that help us keep track of when coherence is lost.
The **Modified-Shared-Invalid** (MSI) protocol builds on the ideas of valid and dirty bits that we have previously seen with caches. Each cache block in each cache is controlled by Finite State Machine that implements the MSI protocol. The Invalid

state is similar to when the valid bit is 0: the data currently stored in a cache block is currently out of date and cannot be used. If a cache block is accessed while in the invalid state, we would get a cache miss. The Shared state is similar to when the valid bit is 1 and the dirty bit is 0: the data in the cache block has never been written to, so we can guarantee that the cache block has coherence with all other caches that are storing the same cache block. Multiple caches can have the same cache block in the Shared state. The Modified state is similar to when the valid bit is 1 and the dirty bit is 1: the data in the cache block has been written to, and we cannot guarantee that its contents match main memory or the other caches. If any cache is storing a cache block in an invalid state, then every other cache must invalidate their copy of that cache block. If a Modified cache block gets replaced, we must copy the contents of that cache block back to memory or to the other caches or both.

Optional for Funsy video about Atomics



Video credits: Geoffrey Herman, Text credits: Geoffrey Herman

Mark as read

☐ (a) I've read this!

Select all possible options that apply.

Save & Grade *Unlimited attempts* Save only