MITOCW | MIT6_004S17_16-02-07-01_300k

Virtual memory allows programs to behave as if they have a larger memory than they actually do.

The way this works is by using virtual addresses, which refer to addresses on disk, in our programs.

The virtual addresses are translated into physical addresses using the page map which is a lookup table that has one entry per virtual page.

The page map knows whether the virtual page is in physical memory and if so it immediately returns the physical page number.

If the page is not in physical memory, then this causes a fault which means that the virtual page must be brought in from disk to physical memory before it can be accessed.

To do this the least recently used (LRU) page in the physical memory is removed to make room for the address that is currently being requested.

The page map is also updated with the new mapping of virtual to physical pages.

Since bringing data to and from disk is an expensive operation, data is moved in chunks.

This makes sense because of the concept of locality which we studied as part of our Caches unit.

The idea is that instructions, or data, that are close to the current address are likely to be accessed as well, so it makes sense to fetch more than one word of data at a time.

This is especially true if the cost of fetching the first word is significantly higher than the cost of fetching adjacent memory locations as is the case with accesses to disk.

So data is moved back and forth from disk in pages.

The size of a page is the same in both virtual and physical memory.

Lets look at an example of how virtual memory is used.

While it is usually the case that the virtual address space is larger than the physical address space, this is not a requirement and in this problem the virtual address space happens to be smaller than the physical address space.

Specifically, virtual addresses are 16 bits long so they can address 2^16 bytes.

Physical addresses are 20 bits long so that means that our physical memory is of size 2^20 bytes.

Our page size is 2^8 bytes or 256 bytes per page.

This means that the 16 bit virtual address consists of 8 bits of page offset and another 8 bits for the virtual page number (or VPN).

The 20 bit physical address consists of the same 8 bit page offset and another 12 bits for the physical page number (or PPN).

The first question we want to consider is what is the size of the page map in this example?

Recall that a page map has 1 entry per virtual page in order to map each virtual page to a physical page.

This means that the number of entries in the page map is 2^8 where 8 is the number of bits in the VPN.

The size of each page map entry is 14 bits, 12 for the PPN, 1 for the dirty bit and 1 for the resident bit.

Suppose that you are told that the page size is doubled in size so that there are now 2^9 bytes per page, but the size of your physical and virtual addresses remain the same.

We would like to determine what effect this change would have on some of the page map attributes.

The first question is how does the size of each page map entry in bits change?

Since the size of a physical address continues to be 20 bits long, then the change in page offset size from 8 to 9 bits implies that the size of the PPN decreased by 1 bit from 12 to 11.

This implies that the size of each page map entry also decreases by 1 bit.

How are the number of entries in the page map affected by the change in page size?

Since the number of entries in a page map is equal to the number of virtual pages, that means that if the size of each page doubled, then we have half as many virtual pages.

This is shown in the size of the VPN which has decreased from 8 to 7 bits.

This also means that the number of entries in the page map have halved in size from 2^8 entries down to 2^7 entries.

How about the number of accesses of the page map that are required to translate a single virtual address?

This parameter does not change as a result of the pages doubling in size.

Suppose we return to our original page size of 256 bytes per page.

We now execute these two lines of code, a load followed by a store operation.

The comment after each instruction shows us the value of the PC when each of the instructions is executed, so it is telling us that the load instruction is at address 0x1FC and the store instruction is at address 0x200.

To execute these two lines of code, we must first fetch each instruction and then perform the data access required by that instruction.

Since our pages are 2^8 bytes long, that means that the bottom 8 bits of our address correspond to the page offset.

Notice that our instruction addresses are specified in hex so 8 bits correspond to the bottom 2 hex characters.

This means that when accessing the LD instruction, the VPN = 1 (which is what remains of our virtual address after removing the bottom 8 bits.) The data accessed by the LD instruction comes from VPN 3.

Next we fetch the store instruction from VPN 2, and finally we store an updated value to VPN 6.

Given the page map shown here, we would like to determine the unique physical addresses that are accessed by this code segment.

Recall that the four virtual addresses that will be accessed are: 0x1FC which is in VPN 1 0x34C which is in VPN 3 0x200 which is in VPN 2 and 0x604 which is in VPN 6.

Assume that all the code and data required to handle page faults is located at physical page 0, your goal is to determine the 5 different physical pages that will get accessed and the order in which they will get accessed by this code segment.

We begin by looking up VPN 1 in our page map.

We see that its resident bit is set to 1.

This means that the virtual page is in physical memory and its PPN is 0x007.

Thus the first physical page that we access is page 0x7, and the first physical address is determined by concatenating the PPN to the page offset.

This results in a physical address of 0x7FC.

Next, we want to load the data at virtual address 0x34C which is in VPN 3.

Looking up VPN 3 in our page map, we find out that its not resident in physical memory.

This means that we need to make room for it by removing the least recently used page from physical memory.

The least recently used page is VPN 2 which maps to PPN 0x602.

Since the dirty bit of our LRU page is 0, that means that we have not done any writes to this page while it was in physical memory so the version in physical memory and on disk are identical.

So to free up physical page 0x602, all we need to do is change the resident bit of VPN 2 to 0 and now we can bring VPN 3 into physical page 0x602.

Recall that the code for handling the page fault is in physical page 0 so the second physical page that we access is page 0.

The updated page map, after handling the page fault, looks like this, where the resident bit for VPN 2 has been set to 0, and PPN 0x602 is now used for VPN 3.

Since this is a LD operation, we are not modifying the page so the dirty bit is set to 0.

The physical address for virtual address 0x34C is now 0x6024C which is now in VPN 0x602.

Next we need to fetch the store instruction from virtual address 0x200 which is in VPN 2.

Since we just removed VPN 2 from physical memory we get another page fault.

This time we will remove the next LRU page from physical memory in order to make room for VPN 2 once again.

In this case, the dirty bit is set to 1 which means that we have written to PPN 0x097 after it was fetched from disk.

This means that the page fault handler will need to first write physical page 0x097 back to virtual page 5 before we can use physical page 0x097 for VPN 2.

After handling the page fault, our updated page map looks like this.

VPN 5 is no longer resident, and instead VPN 2 is resident in physical page 0x097.

In addition, we set the dirty bit to 0 because we have not made any changes to this virtual page.

We now know that virtual address 0x200 maps to physical address 0x09700 after the handling of the page fault.

Finally, we need to perform the store to virtual address 0x604 which is in VPN 6.

Since VPN 6 is resident in physical memory, we can access it at physical page 0x790 as shown in the page map.

This means that virtual address 0x604 maps to physical address 0x79004.

Note that because the dirty bit of VPN 6 was already a 1, we don't need to make any further modifications to the page map as a result of executing the store operation.

If the dirty bit had been a 0, then we would have set it to 1.

So the five physical pages that were accessed by this program are: page 0x7, page 0 for the page faults, page 0x602, page 0x097, and page 0x790.