UCB CS 162 - Operating Systems and Systems Programming "Lecture 13 - Memory 1- Address Translation and Virtual Memory"

Lecture 13 - Memory 1: Address Translation and Virtual Memory

Virtual Address Spaces

An address space is a set of protected memory addresses that are accessible to a program. These are virtual addresses distinct from actual physical memory. Not all processors have virtual address spaces. In order to have virtual address spaces, you need support for address translation from hardware.

The mapping between physical addresses and virtual addresses is done by a unit called the Memory Management Unit (MMU). This mapping lets every process assume that it has the entire RAM to itself, and is able to access any address. Whenever the processes accesses a particular (virtual) address, the MMU maps it to the specific physical address assigned to it, and gives the process data from the actual physical address. However, the process never knows exactly which physical address the data it accessed came from.

Memory Translation

The most important function of the MMU is to be able to allocate non-contiguous chunks of memory to a single process. Without the MMU, in very early operating systems, when a process was loaded into memory, it was assigned a big contiguous chunk of memory. However, allocating contiguous chunks of memory quickly leads to fragmentation of the memory.

To get around this problem, the MMU only allocates contiguous chunks of memory to different "segments" of a process. For example, each process needs memory for its code, static data, heap, and stack. The MMU only allocates contiguous memory for a process's code, static data, heap, and stack, but all of these four parts may be stored in different chunks of physical memory as well as virtual memory.

This approach has another advantage. If we allocate a contiguous chunk of memory for a process' heap, we can map that same chunk of memory to another process' heap and get inter-process communication.

The information about where each segment has been stored is contained in the segment map. For each virtual address, we take the first few bits of the virtual address. These bits are used to index into the segment map and find the base of the chunk of physical memory allocated to this segment. These first few bits are usually the first 2 or 3 bits. Then, these first few bits are replaced by 0s, and the resulting virtual address is considered to be the offset from the base that we got from the segment map. So this offset is added to the base, and we get a physical address.

In the lecture Kubi steps through an assembly program and goes through all the steps I described above in order to get a physical address from a virtual address, around the 1:07:00 timestamp.

This segment table is unique to a process, and when a process is swapped out during a context-switch, this segment table also needs to be swapped out.

Now, if we have so many processes that we run out of room for storing each segment for each process, we can actually use disk to store these segments for a process. This is called swapping, and this lets us use the huge size of the disk as DRAM. However, accessing disk has a huge cost in terms of time, and each disk access takes roughly on the order of 1M cycles. Therefore, this is not really a practical strategy, but only used when there is no other option.

So far, this strategy helped us a little bit with fragmentation. Instead of allocating a huge chunk of contiguous memory to a process, we were able to divide the memory a process requires into different segments, and only allocate those contiguously. However, when the size of the segments keeps growing, especially with the process stack and heap, fragmentation starts coming up again.

Fragmentation can be of two types - external and internal. External fragmentation is the type we have been talking about so far. But internal fragmentation is when we allocate a small(er) chunk of memory to a segment, but the process doesn't need that entire chunk. This leads to internal fragmentation.

In order to better deal with fragmentation (both external and internal), we need a smaller quanta than a segment of a process. We can come up with a smaller quanta, and call it a "page".

Paging

Now each process gets a page table which maps a virtual page number into a physical page number. If you have a virtual address, the last few bits of the address are used to represent an offset into the page. So if a page is of 1024 bytes, we will need to

use the last 10 bits of the virtual address to specify the offset into the page. The remaining bits of the address are used to represent the virtual page number.

Now that you have this virtual address, we use the virtual page number as an index into the page table. The page table gives you the physical page number. You then access that physical page and go to the offset you have, and you get to the address you wanted.

However, this is still not quite what we want. Next lecture extends this idea to better deal with fragmentation.