Notes for 10/26/09

Virtual memory—next step down on illusion of large, fast memory

Advantages to virtual memory:

1. Each process has its own virtual address space—processes can use the same virtual address but have this map to a different frame in physical memory
2. Protection—you cannot modify another process's memory (generally) because you have no pages to that process's memory in your page table
3. Extend our illusion of a big, fast memory—swap seldom used pages to a backing store (e.g., disk), so the OS provides the illusion that you have more physical memory than you really do

Virtual addresses:
Consider 32-bit address where page index bits | page offset bits = 20 | 12.
This means that we have 2^{20} pages.
If memory is byte-addressable, this means that each page is 2^{12} = 4KB.

The operating system maintains a page table for your process in physical memory.

Virtual → physical address translation:
The hardware will walk the page table to translate the virtual addresses used in your instructions into physical addresses. It will use the 20 bits for the page index and look it up in the page table. If the valid bit is set, then it is translated to the corresponding frame in physical memory.

If you cannot find the page or if the valid bit is not set, the hardware will generate a page fault. If this is not handled by the operating system, you will see this as a SIGSEGV.

Processes have different address spaces, but threads of the same process share an address space.

Problem... if we have 2^{20} possible pages, then each page table takes 2^{20} * 4B = 4MB of space.
Solution... use multi-level page table. It is common for 32-bit machines to have two levels and for 64-bit machines to have three levels. For example, with two levels, use the first 10 bits to index the first level, then the other 10 bits to index the second level. This means that we do not need to allocate space for second level tables that we do not need. So now each second level table takes 2^{10} * 4B = 4KB of space.