Multilevel Page Tables

problem: Can't hold all tables in memory.
solution: Page the page tables.
For a 32 bit machine with the max of 4 GB of memory, if the pages are 4KB in size we need 1 million pages, each page table is 32 bits so 4MB. But every program needs its own page table with its own translations and they cannot be stored to disk.
What we do is create one 4KB table with indexes to 1024 other page tables.

Translation proccess

On IA-32 machines.

1. Divide the 32 bit address into the 20 left hand side bits for page indexing and the 12 right hand side bits for the offset within the actual page.
2. If the 20 bits is a TLB (Translation Buffer) hit then, then concatenate the result with the 12 bit offset And you are done.
3. Else divide the 20 bits into 2 10 bits page indexes, the primary which is the page directory which contains mappings to other page tables, and the secondary which points physical locations in memory.
4. Go to the value of the 10 first bits in the page directory, if it contains the address of a valid table (its valid bit is set), load that page table into memory, else you've got a page fault.
5. If the index of the last 10 bits of the first 20 bits is valid then you've found the corrects place in memory with the last 12 bits as its index, else this is a page fault.

This principle can be extended to an arbitrary number of levels of page tables, similar to a shallow tree where each node has a lot of children.
Multilevel page tables can save a lot of memory - the only thing that has to be stored in main memory is the 4KB page directory. The downside is that multilevel page tables has a lot of overhead with multiple disc operation.