Virtual Addresses and the Page Table

Having one huge page table describing every page for the entire memory space would be a very inefficient structure to search when looking for a page.

So, to optimize things, instead of having one large page table, the information is split into multiple page tables, and an index is used to direct the memory manager to the appropriate one for any given request. This index is called the **directory index.**

To translate a 32-bit virtual address into a physical address, it is first split up into three parts:

- directory index (first 10 bits)
- page index (next 10 bits)
- byte offset (last 12 bits).

The three parts of the address are used to navigate through the directory index and the page table to locate the precise address in physical memory.

Figure 7.3 illustrates how the address translation takes place using the three parts of the virtual address.

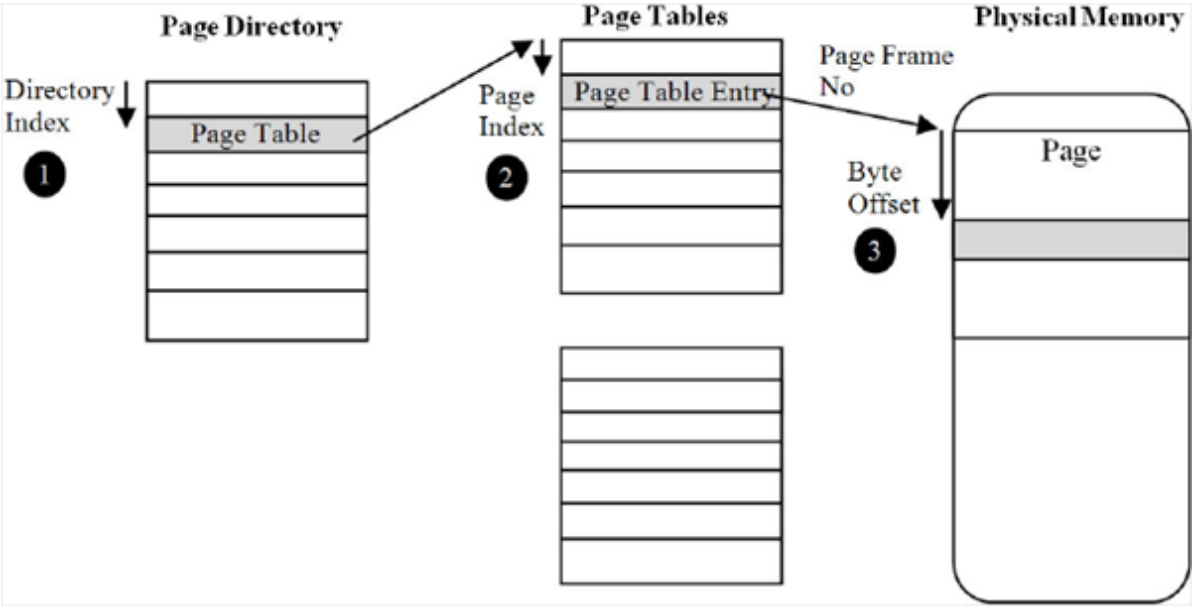

Figure 7.3: Page table structure.

When a virtual address is translated, the first 10 bits are used as an index into the process's directory index table (1). The directory index item at this location contains the address of the page table which the system should use for the next lookup, and the next 10 bits of the virtual address are used as an index into this table (2) which gives the page table entry for the virtual page.

The page table entry is what we ultimately need, because it describes the virtual page and whether the actual page is resident in physical memory. If it is, then the physical memory address (page frame number) is used to find the start of the desired memory address along with the last 12 bits of the virtual address. These last 12 bits are used as an offset from the start of the physical address of the page (3) to give the exact location of the requested memory chunk.

We've just translated a virtual to a physical address!

It's worth pointing out that each process has its own page directory index, the address of which is stored in a specific CPU register so as to be always available when a process thread is running.

The mechanisms for 64-bit processes and 32-bit processes with **physical address extensions (PAE)** are similar, but both involve

more deeply nested table structures to cope with overhead of the larger address space, and the virtual address structure also differs. However, the principle is the same so, in the interests of clarity, I will leave it there.