privileges, or tries to access pages that simply do not exist, results in the raising of a processor exception in a manner analogous to a software interrupt.

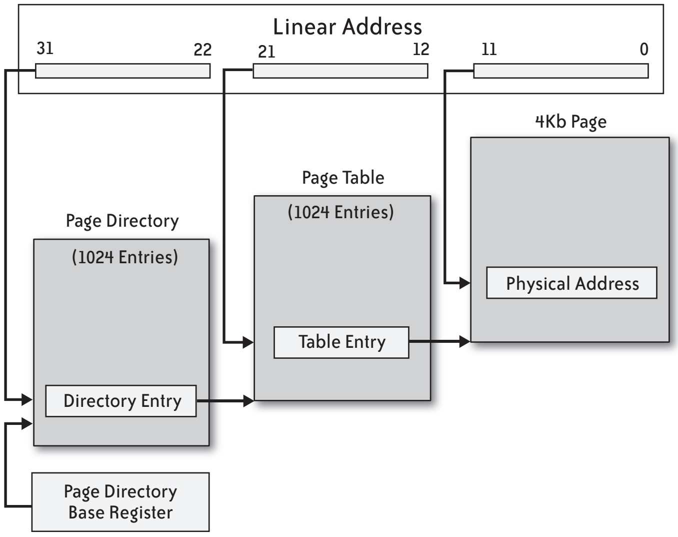

FIGURE 9-4. Paging mechanism of the IA32 architecture (4 KB pages only)

To optimize such a structure, it is clear that our first tactic should be to adopt the approach of the real processor; in other words, we need to have some form of lookup caching. To devise this, we make two key observations:

- The remapping of paging occurs on a granularity of 4 KB. Conveniently, and not by coincidence, this is also the granularity we chose for our physical address space.
- When a protected mode process accesses the memory for read or write, it merely sees a remapping of these 4 KB blocks (while some of the blocks cause processor exceptions). The physical address space is just one possible ordering of the original `Memory` objects (where all the objects are by coincidence in address order) but is otherwise no more significant than any other ordering.

From these observations we see that the most natural form for our cache (i.e., our TLBs) is a duplication of the physical address space structure. The memory pages are mapped according to a new ordering as determined by the page table lookups. On the first request for an address within a given page, a full traversal of the table structure is performed in order to find the matching physical address space memory object. A reference to this object is then deposited at the correct location within the linear address space, and from then on it is cached until we choose to clear it.

To solve the problem of read/write and user/supervisor, we make a tactical decision to sacrifice some memory overhead for speed. We produce a linear address space for each combination: