directory with a single entry, a pointer to its one and only page table. In this way, the overhead for short segments is only two pages, instead of the million pages that would be needed in a one-level page table.

To avoid making repeated references to memory, the Pentium, like MULTICS, has a small TLB that directly maps the most recently used *Dir–Page* combinations onto the physical address of the page frame. Only when the current combination is not present in the TLB is the mechanism of Fig. 3-42 actually carried out and the TLB updated. As long as TLB misses are rare, performance is good.

It is also worth noting that if some application does not need segmentation but is content with a single, paged, 32-bit address space, that model is possible. All the segment registers can be set up with the same selector, whose descriptor has *Base* = 0 and *Limit* set to the maximum. The instruction offset will then be the linear address, with only a single address space used—in effect, normal paging. In fact, all current operating systems for the Pentium work this way. OS/2 was the only one that used the full power of the Intel MMU architecture.

All in all, one has to give credit to the Pentium designers. Given the conflicting goals of implementing pure paging, pure segmentation, and paged segments, while at the same time being compatible with the 286, and doing all of this efficiently, the resulting design is surprisingly simple and clean.

Although we have covered the complete architecture of the Pentium virtual memory, albeit briefly, it is worth saying a few words about protection, since this subject is intimately related to the virtual memory. Just as the virtual memory scheme is closely modeled on MULTICS, so is the protection system. The Pentium supports four protection levels, with level 0 being the most privileged and level 3 the least. These are shown in Fig. 3-43. At each instant, a running program is at a certain level, indicated by a 2-bit field in its PSW. Each segment in the system also has a level.

As long as a program restricts itself to using segments at its own level, everything works fine. Attempts to access data at a higher level are permitted. Attempts to access data at a lower level are illegal and cause traps. Attempts to call procedures at a different level (higher or lower) are allowed, but in a carefully controlled way. To make an interlevel call, the CALL instruction must contain a selector instead of an address. This selector designates a descriptor called a **call gate**, which gives the address of the procedure to be called. Thus it is not possible to jump into the middle of an arbitrary code segment at a different level. Only official entry points may be used. The concepts of protection levels and call gates were pioneered in MULTICS, where they were viewed as **protection rings**.

A typical use for this mechanism is suggested in Fig. 3-43. At level 0, we find the kernel of the operating system, which handles I/O, memory management, and other critical matters. At level 1, the system call handler is present. User programs may call procedures here to have system calls carried out, but only a specific and protected list of procedures may be called. Level 2 contains library procedures, possibly shared among many running programs. User programs may call

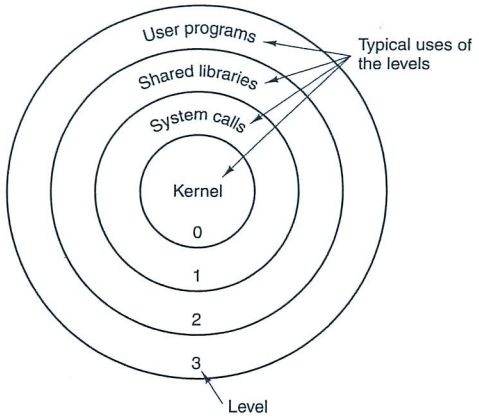

Figure 3-43. Protection on the Pentium.

these procedures and read their data, but they may not modify them. Finally, user programs run at level 3, which has the least protection.

Traps and interrupts use a mechanism similar to the call gates. They, too, reference descriptors, rather than absolute addresses, and these descriptors point to specific procedures to be executed. The *Type* field in Fig. 3-40 distinguishes between code segments, data segments, and the various kinds of gates.

3.8 RESEARCH ON MEMORY MANAGEMENT

Memory management, especially paging algorithms, was once a fruitful area for research, but most of that seems to have largely died off, at least for general-purpose systems. Most real systems tend to use some variation on clock, because it is easy to implement and relatively effective. One recent exception, however, is a redesign of the 4.4 BSD virtual memory system (Cranor and Parulkar, 1999).

There is still research going on concerning paging in newer kinds of systems though. For example, cell phones and PDAs have become small PCs, and many of them page RAM to "disk," only disk on a cell phone is flash memory, which has different properties than a rotating magnetic disk. Some recent work is reported by (In et al., 2007; Joo et al., 2006; and Park et al., 2004a). Park et al. (2004b) have also looked at energy-aware demand paging in mobile devices.

Research is also taking place on modeling paging performance (Albers et al., 2002; Burton and Kelly, 2003; Cascaval et al., 2005; Panagiotou and Souza, 2006; and Peserico, 2003). Also of interest is memory management for multimedia systems (Dasigenis et al., 2001; Hand, 1999) and real-time systems (Pizlo and Vitek, 2006).