CS162, Spring 2004 **Discussion #8** **Amir Kamil**
UC Berkeley **3/11/04**

Topics: Paging, Multilevel Paging

1 Announcements

- Group evaluations should be done a day after project turned in. Since the webpage said two days, we will give you the extra day for phase 1.
- Midterm Thursday, March 18th, 2004, 5:30-7:30pm, 155 Dwinelle
- Project 2 initial design doc due FRIDAY, March 12, 2004 @ 11:59pm
- Midterm review 3/17 in class
- proj1-code grades have been entered, stats online

2 Paging

Recall that in segmentation, external fragmentation, or holes between segments in memory, is a large problem. Complex memory allocation algorithms are required to reduce this problem, and in some cases whole segments must be copied to obtain a large enough chunk of free contiguous memory.

The problems of segmentation arise mostly from the fact that segments can be variable size. So in order to solve these problems, we can force segments to be of a fixed size. Such a scheme is called *paging*, and the segments are now called *pages*. Pages are usually about 4KB in size.

In paging, a virtual address is now divided into two pieces, a virtual page number and the offset in the page. Since pages are 4KB each, the offset uses 12 bits ($2^{12} = 4096$), and the page number uses 20 bits. In order to translate this into a physical address, each process has a page table that maps virtual pages to physical pages. The memory management unit looks up the page table entry for the virtual page and replaces the virtual page number with the corresponding physical page while keeping the offset in order to obtain a physical address, as shown in figure 1. A page table entry also has some extra information, such as protection, used, and valid bits. A reference to an invalid page is called a *page fault*, and we will see later what can cause this. Table 1 gives some examples of address translation in a 16-bit virtual, 20-bit physical address space with 4KB pages.

Paging has multiple advantages over segmentation. We no longer need a complex memory allocation algorithm, since any free physical page can be used when space is allocated. Program sections no longer need to be contiguous, and there is no external fragmentation. Sharing is easy, since whole pages can be shared, and we will see later that pages can actually be moved to disk when they are not in use.

However, paging also introduces new problems. A process can only allocate memory in multiples of the page size, so it can own a page of which it only uses a small amount. This is called *internal fragmentation*. In order to reduce this problem, small page sizes are used, but then the page table size is large. Consider a 32-bit address space with 4KB pages. There are 20 bits of page numbers, or $2^{20} = 1$M total entries in the page table. Each page table entry must hold 20 bits for the corresponding physical page number, and a few more bits for protection and accounting information, so about 32 bits total. Thus the page table size is 1M $\cdot$32 bits = 4MB, all of which must be in physical memory. Since each process has its own page table, 4MB of memory per process is required in paging. With many processes in a system, the page tables can potentially take up a large chunk of physical memory.

3 Multilevel Paging

One solution to the large memory requirements of page tables is to use *multilevel paging*. In this scheme, a virtual address is divided into three or more sections, with all but the last section being page numbers in different page tables, and the last one being the offset. In two-level paging, the first section is the page

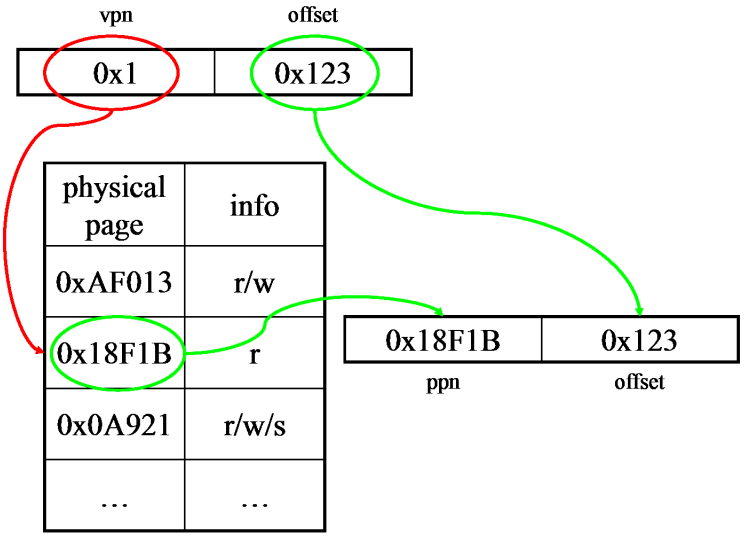

Figure 1: Translation procedure in paging for a 32-bit address space with 4KB pages.

Virtual page number	Physical page number	Valid
0	0x14	Y
1	0x50	Y
2	0x52	Y
3	0xFE	N
4	0xA4	Y
5	0x26	Y
6	0x88	Y
7	0x68	Y
8	0xD4	N
9	0x18	Y
...	...	...

Virtual address	Physical address
0x156D	0x5056D
0x6400	0x88400
0x8888	Page fault.
0x7888	0x68888
0x23F4	0x523F4
0x3A6B	Page fault.

Table 1: Examples of address translation, using 4KB pages in a 16-bit virtual, 20-bit physical address space.

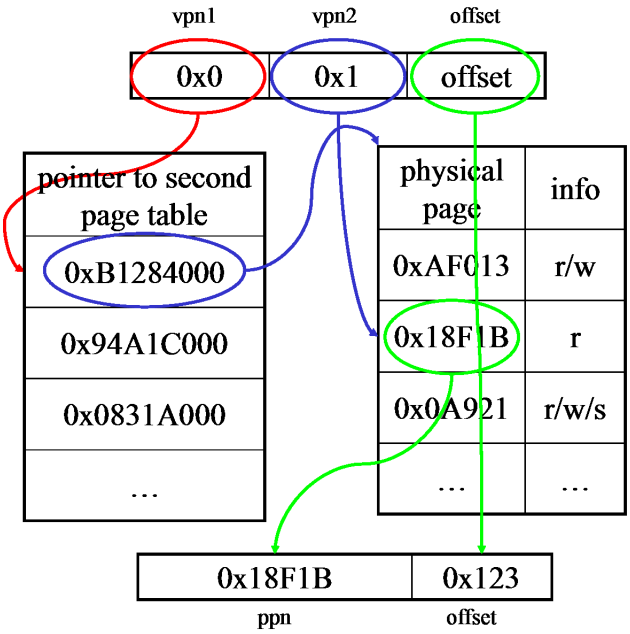

Figure 2: Translation procedure in multilevel paging for a 32-bit address space with 4KB pages

number in a top-level page table, and is used to lookup a second-level page table. The second section is the page number in the second-level table, and is used to lookup the physical page number. Figure 2 shows the translation scheme for two-level paging in a 32-bit address space, with 10 bits for each virtual page number. Note that the top-level table holds a pointer to the corresponding second-level table, but the second-level table holds a 20-bit physical page number.

How much space is required for the page tables in this scheme? We now have a top-level table with $2^{10} = 1024$ entries, so it takes 4KB of space. There are 1024 second-level tables, each with 1024 entries, so they take 4MB total. So this scheme takes 4KB more than single-level paging. However, not all the second-level tables need to be resident in memory, and page tables are not needed for unmapped virtual memory. Thus the total usage is actually lower, and only the top-level table must be in memory at all times, requiring only 4KB (exactly one page!) of physical memory per process.

A minor complication arises in multilevel translation: should the address of the second-level page table in the top-level table be a virtual or a physical address? A virtual address adds an extra translation that needs to be done, while a physical address has problems dealing with page tables that are not in memory. Generally, the top-level table contains physical addresses, but the least significant bit of each entry indicates whether the page table is present in memory or not. If not present, then the entry would contain an OS-specific pointer (such as a disk block address) used to record where the page table is stored on disk.

Multilevel translation schemes can use segmentation as well, and a mixture of segmentation and paging. For example, the top-level table could be a segment table, and the second-level a page table. The translation scheme then mimics the segmentation scheme for the first level and the paging scheme for the second.

While the memory requirements are lower for multilevel tables, the extra levels require more memory accesses to translate an address. This problem is alleviated by caching translations, in a *translation lookaside buffer (TLB)*.