Initial value of $D_i$ is 0
After an operation, the value of $D_i$ is updated as follows

**(I)** if the next operation is a block allocate request:
    if there is any free block, select one to allocate
        if the selected block is locally free
            then $D_i := D_i + 2$
            else $D_i := D_i + 1$
    otherwise
        first get two blocks by splitting a larger one into two (recursive operation)
        allocate one and mark the other locally free
        $D_i$ remains unchanged (but $D$ may change for other block sizes because
                of the recursive call)

**(II)** if the next operation is a block free request
    Case $D_i \geqslant 2$
        mark it locally free and free it locally
        $D_i = 2$
    Case $D_i = 1$
        mark it globally free and free it globally; coalesce if possible
        $D_i = 0$
    Case $D_i = 0$
        mark it globally free and free it globally; coalesce if possible
        select one locally free block of size $2^i$ and free it globally; coalesce if possible
        $D_i := 0$

**Figure 8.24** **Lazy Buddy System Algorithm**

## 8.4 LINUX MEMORY MANAGEMENT

Linux shares many of the characteristics of the memory management schemes of other UNIX implementations but has its own unique features. Overall, the Linux memory management scheme is quite complex [DUBE98]. In this section, we give a brief overview of the two main aspects of Linux memory management: process virtual memory and kernel memory allocation.

### Linux Virtual Memory

***VIRTUAL MEMORY ADDRESSING*** Linux makes use of a three-level page table structure, consisting of the following types of tables (each individual table is the size of one page):

- **Page directory:** An active process has a single page directory that is the size of one page. Each entry in the page directory points to one page of the page middle directory. The page directory must be in main memory for an active process.

- **Page middle directory:** The page middle directory may span multiple pages. Each entry in the page middle directory points to one page in the page table.
- **Page table:** The page table may also span multiple pages. Each page table entry refers to one virtual page of the process.

To use this three-level page table structure, a virtual address in Linux is viewed as consisting of four fields (Figure 8.25). The leftmost (most significant) field is used as an index into the page directory. The next field serves as an index into the page middle directory. The third field serves as an index into the page table. The fourth field gives the offset within the selected page of memory.

The Linux page table structure is platform independent and was designed to accommodate the 64-bit Alpha processor, which provides hardware support for three levels of paging. With 64-bit addresses, the use of only two levels of pages on the Alpha would result in very large page tables and directories. The 32-bit Pentium/x86 architecture has a two-level hardware paging mechanism. The Linux software accommodates the two-level scheme by defining the size of the page middle directory as one. Note that all references to an extra level of indirection are optimized away at compile time, not at run time. Therefore, there is no performance overhead for using generic three-level design on platforms which support only two levels in hardware.

***PAGE ALLOCATION*** To enhance the efficiency of reading in and writing out pages to and from main memory, Linux defines a mechanism for dealing with contiguous blocks of pages mapped into contiguous blocks of page frames. For this purpose, the buddy system is used. The kernel maintains a list of contiguous page frame groups of fixed size; a group may consist of 1, 2, 4, 8, 16, or 32 page frames. As pages

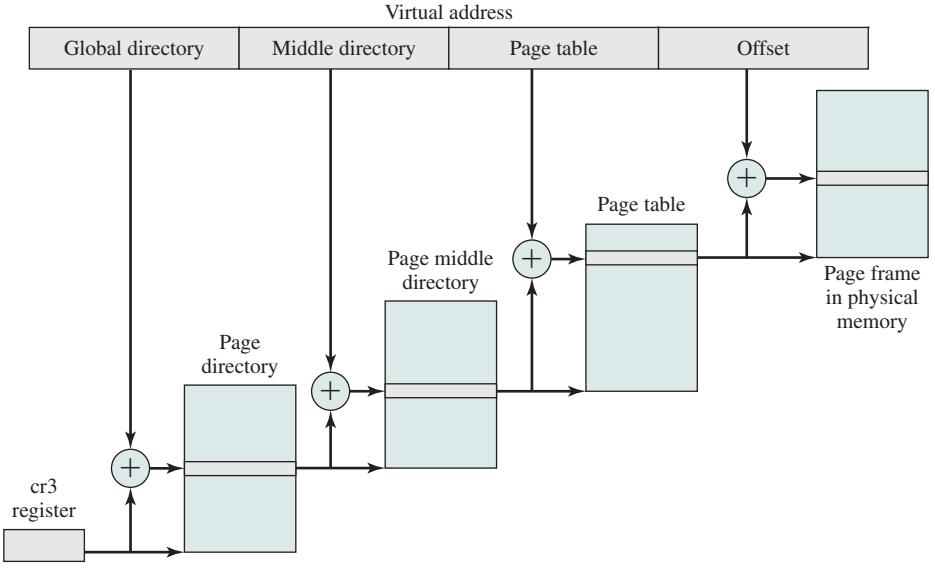


**Figure 8.25 Address Translation in Linux Virtual Memory Scheme**

are allocated and deallocated in main memory, the available groups are split and merged using the buddy algorithm.

***PAGE REPLACEMENT ALGORITHM*** The Linux page replacement algorithm is based on the clock algorithm described in Section 8.2 (see Figure 8.16). In the simple clock algorithm, a use bit and a modify bit are associated with each page in main memory. In the Linux scheme, the use bit is replaced with an 8-bit age variable. Each time that a page is accessed, the age variable is incremented. In the background, Linux periodically sweeps through the global page pool and decrements the age variable for each page as it rotates through all the pages in main memory. A page with an age of 0 is an "old" page that has not been referenced in some time and is the best candidate for replacement. The larger the value of age, the more frequently a page has been used in recent times and the less eligible it is for replacement. Thus, the Linux algorithm is a form of least frequently used policy.

### Kernel Memory Allocation

The Linux kernel memory capability manages physical main memory page frames. Its primary function is to allocate and deallocate frames for particular uses. Possible owners of a frame include user-space processes (i.e., the frame is part of the virtual memory of a process that is currently resident in real memory), dynamically allocated kernel data, static kernel code, and the page cache.[7]

The foundation of kernel memory allocation for Linux is the page allocation mechanism used for user virtual memory management. As in the virtual memory scheme, a buddy algorithm is used so that memory for the kernel can be allocated and deallocated in units of one or more pages. Because the minimum amount of memory that can be allocated in this fashion is one page, the page allocator alone would be inefficient because the kernel requires small short-term memory chunks in odd sizes. To accommodate these small chunks, Linux uses a scheme known as *slab allocation* [BONW94] within an allocated page. On a Pentium/x86 machine, the page size is 4 Kbytes, and chunks within a page may be allocated of sizes 32, 64, 128, 252, 508, 2,040, and 4,080 bytes.

The slab allocator is relatively complex and is not examined in detail here; a good description can be found in [VAHA96]. In essence, Linux maintains a set of linked lists, one for each size of chunk. Chunks may be split and aggregated in a manner similar to the buddy algorithm and moved between lists accordingly.

## 8.5 WINDOWS MEMORY MANAGEMENT

The Windows virtual memory manager controls how memory is allocated and how paging is performed. The memory manager is designed to operate over a variety of platforms and to use page sizes ranging from 4 Kbytes to 64 Kbytes. Intel

[7]The page cache has properties similar to a disk buffer, described in this chapter, as well as a disk cache, described in Chapter 11. We defer a discussion of the Linux page cache to Chapter 11.

and AMD64 platforms have 4 Kbytes per page and Intel Itanium platforms have 8 Kbytes per page.

## Windows Virtual Address Map

On 32-bit platforms, each Windows user process sees a separate 32-bit address space, allowing 4 Gbytes of virtual memory per process. By default, half of this memory is reserved for the OS, so each user actually has 2 Gbytes of available virtual address space and all processes share most of the upper 2 Gbytes of system space when running in kernel-mode. Large memory intensive applications, on both clients and servers, can run more effectively using 64-bit Windows. Other than netbooks, most modern PCs use the AMD64 processor architecture which is capable of running as either a 32-bit or 64-bit system.

Figure 8.26 shows the default virtual address space seen by a normal 32-bit user process. It consists of four regions:

- **0x00000000 to 0x0000FFFF**: Set aside to help programmers catch NULL-pointer assignments.
- **0x00010000 to 0x7FFEFFFF**: Available user address space. This space is divided into pages that may be loaded into main memory.

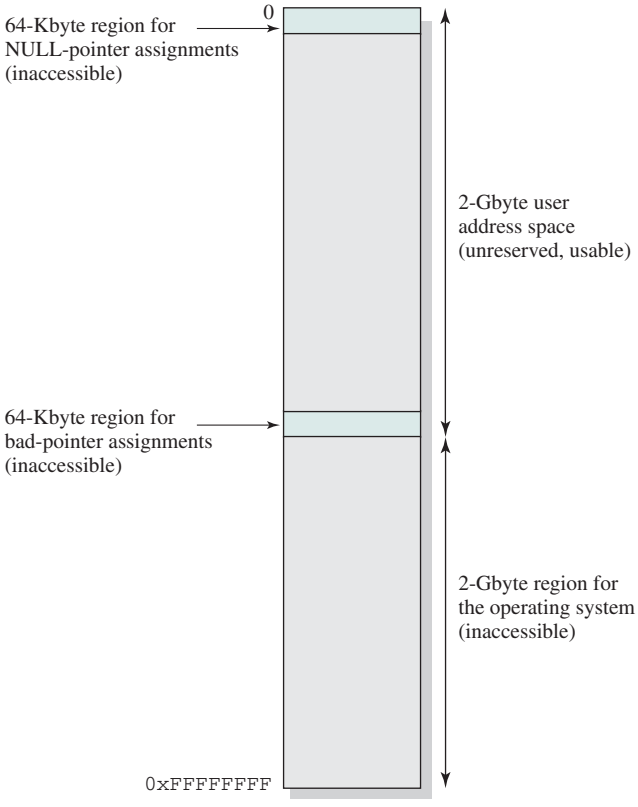


**Figure 8.26 Windows Default 32-Bit Virtual Address Space**

- **0x7FFF0000 to 0x7FFFFFFF**: A guard page inaccessible to the user. This page makes it easier for the OS to check on out-of-bounds pointer references.
- **0x80000000 to 0xFFFFFFFF**: System address space. This 2-Gbyte process is used for the Windows Executive, Kernel, HAL, and device drivers.
- On 64-bit platforms, 8 Tbytes of user address space is available in Windows 7.

## Windows Paging

When a process is created, it can in principle make use of the entire user space of almost 2 Gbytes (or 8 Tbytes on 64-bit Windows). This space is divided into fixed-size pages, any of which can be brought into main memory, but the OS manages the addresses in contiguous regions allocated on 64-Kbyte boundaries. A region can be in one of three states:

- **Available:** addresses not currently used by this process.
- **Reserved:** addresses that the virtual memory manager has set aside for a process so they cannot be allocated to another use (e.g., saving contiguous space for a stack to grow).
- **Committed:** addresses that the virtual memory manager has initialized for use by the process to access virtual memory pages. These pages can reside either on disk or in physical memory. When on disk they can be either kept in files (mapped pages) or occupy space in the paging file (i.e., the disk file to which it writes pages when removing them from main memory).

The distinction between reserved and committed memory is useful because it (1) reduces the amount of total virtual memory space needed by the system, allowing the page file to be smaller; and (2) allows programs to reserve addresses without making them accessible to the program or having them charged against their resource quotas.

The resident set management scheme used by Windows is variable allocation, local scope (see Table 8.5). When a process is first activated, it is assigned data structures to manage its working set. As the pages needed by the process are brought into physical memory the memory manager uses the data structures to keep track of the pages assigned to the process. Working sets of active processes are adjusted using the following general conventions:

- When main memory is plentiful, the virtual memory manager allows the resident sets of active processes to grow. To do this, when a page fault occurs, a new physical page is added to the process but no older page is swapped out, resulting in an increase of the resident set of that process by one page.
- When memory becomes scarce, the virtual memory manager recovers memory for the system by removing less recently used pages out of the working sets of active processes, reducing the size of those resident sets.
- Even when memory is plentiful, Windows watches for large processes that are rapidly increasing their memory usage. The system begins to remove

pages that have not been recently used from the process. This policy makes the system more responsive because a new program will not suddenly cause a scarcity of memory and make the user wait while the system tries to reduce the resident sets of the processes that are already running.

## 8.6 SUMMARY

To use the processor and the I/O facilities efficiently, it is desirable to maintain as many processes in main memory as possible. In addition, it is desirable to free programmers from size restrictions in program development.

The way to address both of these concerns is virtual memory. With virtual memory, all address references are logical references that are translated at run time to real addresses. This allows a process to be located anywhere in main memory and for that location to change over time. Virtual memory also allows a process to be broken up into pieces. These pieces need not be contiguously located in main memory during execution and, indeed, it is not even necessary for all of the pieces of the process to be in main memory during execution.

Two basic approaches to providing virtual memory are paging and segmentation. With paging, each process is divided into relatively small, fixed-size pages. Segmentation provides for the use of pieces of varying size. It is also possible to combine segmentation and paging in a single memory management scheme.

A virtual memory management scheme requires both hardware and software support. The hardware support is provided by the processor. The support includes dynamic translation of virtual addresses to physical addresses and the generation of an interrupt when a referenced page or segment is not in main memory. Such an interrupt triggers the memory management software in the OS.

A number of design issues relate to OS support for memory management:

- **Fetch policy:** Process pages can be brought in on demand, or a prepaging policy can be used, which clusters the input activity by bringing in a number of pages at once.
- **Placement policy:** With a pure segmentation system, an incoming segment must be fit into an available space in memory.
- **Replacement policy:** When memory is full, a decision must be made as to which page or pages are to be replaced.
- **Resident set management:** The OS must decide how much main memory to allocate to a particular process when that process is swapped in. This can be a static allocation made at process creation time, or it can change dynamically.
- **Cleaning policy:** Modified process pages can be written out at the time of replacement, or a precleaning policy can be used, which clusters the output activity by writing out a number of pages at once.
- **Load control:** Load control is concerned with determining the number of processes that will be resident in main memory at any given time.