takes the data byte and then clears the bit in the control register to signal that it is ready for the next byte. Then the CPU can transfer the next byte. If the CPU uses polling to watch the control bit, constantly looping to see whether the device is ready, this method of operation is called **programmed I/O (PIO)**. If the CPU does not poll the control bit, but instead receives an interrupt when the device is ready for the next byte, the data transfer is said to be **interrupt driven**.

9.8 Allocating Kernel Memory

When a process running in user mode requests additional memory, pages are allocated from the list of free page frames maintained by the kernel. This list is typically populated using a page-replacement algorithm such as those discussed in Section 9.4 and most likely contains free pages scattered throughout physical memory, as explained earlier. Remember, too, that if a user process requests a single byte of memory, internal fragmentation will result, as the process will be granted an entire page frame.

Kernel memory is often allocated from a free-memory pool different from the list used to satisfy ordinary user-mode processes. There are two primary reasons for this:

1. The kernel requests memory for data structures of varying sizes, some of which are less than a page in size. As a result, the kernel must use memory conservatively and attempt to minimize waste due to fragmentation. This is especially important because many operating systems do not subject kernel code or data to the paging system.
2. Pages allocated to user-mode processes do not necessarily have to be in contiguous physical memory. However, certain hardware devices interact directly with physical memory—without the benefit of a virtual memory interface—and consequently may require memory residing in physically contiguous pages.

In the following sections, we examine two strategies for managing free memory that is assigned to kernel processes: the "buddy system" and slab allocation.

9.8.1 Buddy System

The buddy system allocates memory from a fixed-size segment consisting of physically contiguous pages. Memory is allocated from this segment using a **power-of-2 allocator**, which satisfies requests in units sized as a power of 2 (4 KB, 8 KB, 16 KB, and so forth). A request in units not appropriately sized is rounded up to the next highest power of 2. For example, a request for 11 KB is satisfied with a 16-KB segment.

Let's consider a simple example. Assume the size of a memory segment is initially 256 KB and the kernel requests 21 KB of memory. The segment is initially divided into two **buddies**—which we will call A_L and A_R—each 128 KB in size. One of these buddies is further divided into two 64-KB buddies—B_L and B_R. However, the next-highest power of 2 from 21 KB is 32 KB so either B_L or B_R is again divided into two 32-KB buddies, C_L and C_R. One of these

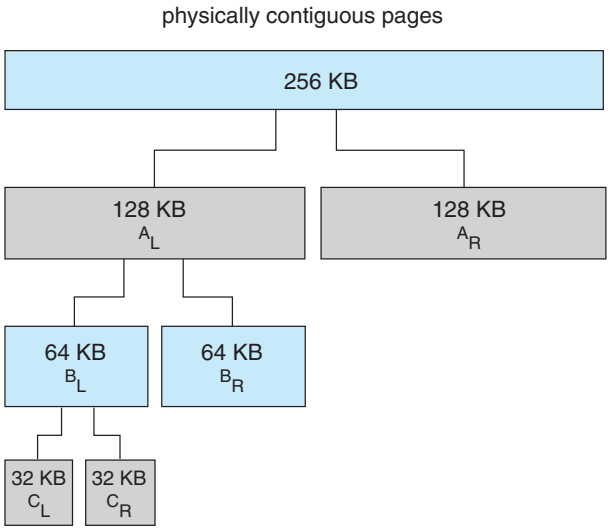

Figure 9.26 Buddy system allocation.

buddies is used to satisfy the 21-KB request. This scheme is illustrated in Figure 9.26, where C_L is the segment allocated to the 21-KB request.

An advantage of the buddy system is how quickly adjacent buddies can be combined to form larger segments using a technique known as **coalescing**. In Figure 9.26, for example, when the kernel releases the C_L unit it was allocated, the system can coalesce C_L and C_R into a 64-KB segment. This segment, B_L, can in turn be coalesced with its buddy B_R to form a 128-KB segment. Ultimately, we can end up with the original 256-KB segment.

The obvious drawback to the buddy system is that rounding up to the next highest power of 2 is very likely to cause fragmentation within allocated segments. For example, a 33-KB request can only be satisfied with a 64-KB segment. In fact, we cannot guarantee that less than 50 percent of the allocated unit will be wasted due to internal fragmentation. In the following section, we explore a memory allocation scheme where no space is lost due to fragmentation.

9.8.2 Slab Allocation

A second strategy for allocating kernel memory is known as **slab allocation**. A **slab** is made up of one or more physically contiguous pages. A **cache** consists of one or more slabs. There is a single cache for each unique kernel data structure —for example, a separate cache for the data structure representing process descriptors, a separate cache for file objects, a separate cache for semaphores, and so forth. Each cache is populated with **objects** that are instantiations of the kernel data structure the cache represents. For example, the cache representing semaphores stores instances of semaphore objects, the cache representing process descriptors stores instances of process descriptor objects, and so forth. The relationship among slabs, caches, and objects is shown in Figure 9.27. The figure shows two kernel objects 3 KB in size and three objects 7 KB in size, each stored in a separate cache.