if the working directory is `C:\My Documents\letters`, the following relative path names are also valid:

```
..\landscape.jpg
..\csc111\proj2.java
..\..\WINDOWS\Drivers\E55IC.ICM
..\..\Program Files\WinZip
```

UNIX systems work essentially the same way. Using the directory tree in Figure 11.5, and assuming that the current working directory is `/home/jones`, the following are valid relative path names:

```
utilities/combine
../smith/reports
../../dev/ttyE71
../../usr/man/man1/ls.1.gz
```

Most operating systems allow the user to specify a set of paths that are searched (in a specific order) to help resolve references to executable programs. Often that set of paths is specified using an operating system variable called `PATH`, which holds a string that contains several absolute path names. Suppose, for instance, that user `jones` (from Figure 11.5) has a set of utility programs that he uses from time to time. They are stored in the directory `/home/jones/utilities`. When that path is added to the `PATH` variable, it becomes a standard location used to find programs that `jones` attempts to execute. Therefore, no matter what the current working directory is, when `jones` executes the `printall` program (just the name by itself), it is found in his utilities directory.

11.3 Disk Scheduling

The most important hardware device used as secondary memory is the magnetic disk drive. File systems stored on these drives must be accessed in an efficient manner. It turns out that transferring data to and from secondary memory is the worst bottleneck in a general computer system.

Recall from Chapter 10 that the speed of the CPU and the speed of main memory are much faster than the speed of data transfer to and from secondary memory such as a magnetic disk. That's why a process that must perform I/O to disk is made to wait while that data is transferred, to give another process a chance to use the CPU.

Because secondary I/O is the slowest aspect of a general computer system, the techniques for accessing data on a disk drive are of crucial importance to file systems. As a computer deals with multiple processes over a period of time, requests to access the disk accumulate. The technique that the operating

Disk scheduling The act of deciding which outstanding requests for disk I/O to satisfy first

system uses to determine which requests to satisfy first is called disk scheduling. We examine several specific disk-scheduling algorithms in this section.

Recall from Chapter 5 that a magnetic disk drive is organized as a stack of platters, where each platter is divided into tracks, and each track is divided into sectors. The set of corresponding tracks on all platters is called a cylinder. Figure 11.6 revisits the disk drive depicted in Chapter 5 to remind you of this organization.

Of primary importance to us in this discussion is the fact that the set of read/write heads hovers over a particular cylinder along all platters at any given point in time. The *seek time* is the amount of time it takes for the heads to reach the appropriate cylinder. The *latency* is the additional time it takes the platter to rotate into the proper position so that the data can be read or written. Seek time is the more restrictive of these two parameters and, therefore, is the primary issue dealt with by the disk-scheduling algorithms.

At any point in time, a disk drive may have a set of outstanding requests that must be satisfied. For now, we consider only the cylinder (the parallel concentric circles) to which the requests refer. A disk may have thousands of cylinders. To keep things simple, let's also assume a range of 110 cylinders. Suppose at a particular time the following cylinder requests have been made, in this order:

49, 91, 22, 61, 7, 62, 33, 35

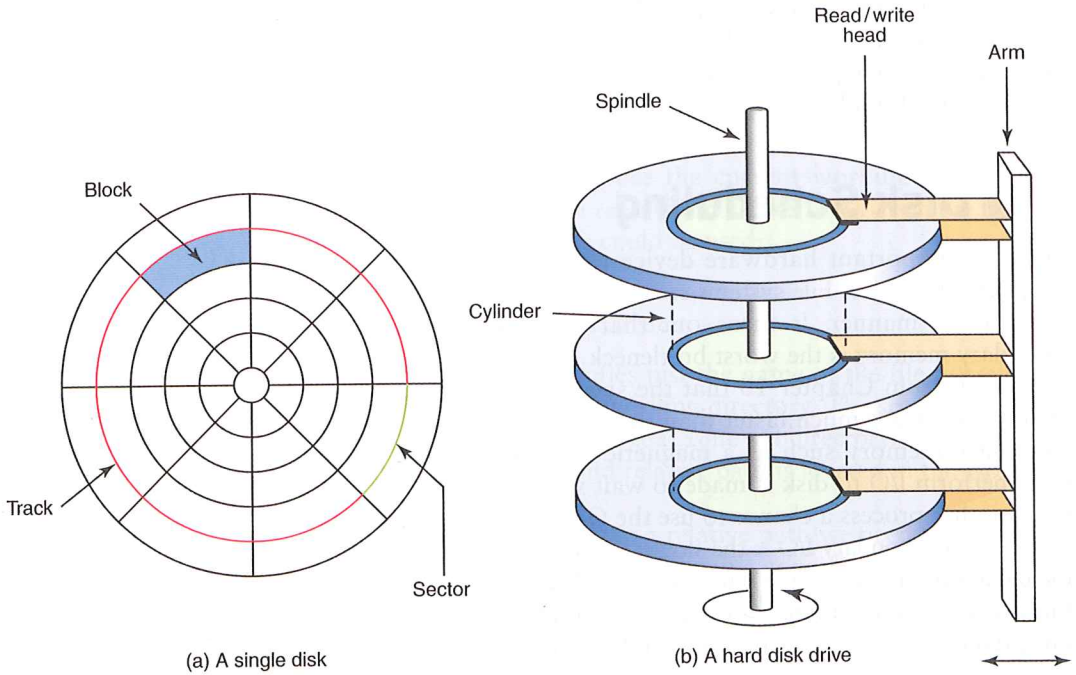

FIGURE 11.6 A magnetic disk drive

Suppose also that the read/write heads are currently at cylinder 26. Now a question arises: To which cylinder should the disk heads move next? Different algorithms produce different answers to this question.

First-Come, First-Served Disk Scheduling

In Chapter 10 we examined a CPU scheduling algorithm called *first come, first served* (FCFS). An analogous algorithm can be used for disk scheduling. It is one of the easiest to implement, though not usually the most efficient.

In FCFS, we process the requests in the order they arrive, without regard to the current position of the heads. Therefore, under a FCFS algorithm, the heads move from cylinder 26 (the current position) to cylinder 49. After the request for cylinder 49 is satisfied (that is, the data is read or written), the heads move from 49 to 91. After processing the request at 91, the heads move to cylinder 22. Processing continues in this manner, satisfying requests in the order that they were received.

Note that at one point the heads move from cylinder 91 all the way back to cylinder 22, during which they pass over several cylinders whose requests are currently pending.

Shortest-Seek-Time-First Disk Scheduling

The *shortest-seek-time-first* (SSTF) disk-scheduling algorithm moves the heads by the minimum amount necessary to satisfy any pending request. This approach could potentially result in the heads changing directions after each request is satisfied.

Let's process our hypothetical situation using this algorithm. From our starting point at cylinder 26, the closest cylinder among all pending requests is 22. So, ignoring the order in which the requests came, the heads move to cylinder 22 to satisfy that request. From 22, the closest request is for cylinder 33, so the heads move there. The closest unsatisfied request to 33 is at cylinder 35. The distance to cylinder 49 is now the smallest, so the heads move there next. Continuing that approach, the rest of the cylinders are visited in the following order: 49, 61, 62, 91, 7.

This approach does not guarantee the smallest overall head movement, but it generally offers an improvement in performance over the FCFS algorithm. However, a major problem can arise with this approach. Suppose requests for cylinders continue to build up while existing ones are being satisfied. And suppose those new requests are always closer to the current position than an earlier request. It is theoretically possible that the early request never gets processed because requests keep arriving that take priority. This situation is called *starvation*. By contrast, FCFS disk scheduling cannot suffer from starvation.

SCAN Disk Scheduling

A classic example of algorithm analysis in computing comes from the way an elevator is designed to visit floors that have people waiting. In general, an elevator moves from one extreme to the other (say, the top of the building to the bottom), servicing requests as appropriate. It then travels from the bottom to the top, servicing those requests.

The *SCAN* disk-scheduling algorithm works in a similar way, except that instead of moving up and down, the read/write heads move in toward the spindle, then out toward the platter edge, then back toward the spindle, and so forth.

Let's use this algorithm to satisfy our set of requests. Unlike in the other approaches, though, we need to decide which way the heads are moving initially. Let's assume they are moving toward the lower cylinder values (and are currently at cylinder 26).

As the read/write heads move from cylinder 26 toward cylinder 1, they satisfy the requests at cylinders 22 and 7 (in that order). After reaching cylinder 1, the heads reverse direction and move all the way out to the other extreme. Along the way, they satisfy the following requests, in order: 33, 35, 49, 61, 62, 91.

New requests are not given any special treatment under this scheme. They may or may not be serviced before earlier requests—it depends on the current location of the heads and direction in which they are moving. If a new request arrives just before the heads reach that cylinder, it is processed right away. If it arrives just after the heads move past that cylinder, it must wait for the heads to return. There is no chance for starvation because each cylinder is processed in turn.

Some variations on this algorithm can improve its performance. For example, a request at the edge of the platter may have to wait for the heads to move almost all the way to the spindle and all the way back. To improve the average wait time, the Circular SCAN algorithm treats the disk as if it were a ring and not a disk. That is, when it reaches one extreme, the heads return all the way to the other extreme without processing requests.

Another variation is to minimize the extreme movements at the spindle and at the edge of the platter. Instead of going to the edge, the heads move only as far out (or in) as the outermost (or innermost) request. Before moving on to tackle the next request, the list of pending requests is examined to see whether movement in the current direction is warranted. This variation is referred to as the *LOOK* disk-scheduling algorithm, because it looks ahead to see whether the heads should continue in the current direction.

Keeping the elderly at home

Many new technologies are being developed to make it easier for the elderly to continue living independent lives at home. One example is the eNeighbor® system, a system with 12 kinds of sensors (e.g., bed, toilet flush, home away, open/closed) and an optional Web camera. Wireless sensors can detect changes in a person's habits (if a person falls down and does not move in his or her regular pattern, for example) and will relay that information to a central monitoring system. An operator then calls the house to ask if the patient is okay. If no response is received, more calls are made to family, neighbors, or the 911 emergency system. While the system is not covered under all insurance plans, the cost is still much less than that of a nursing home, and it allows the elderly the freedom to live at home.