...). They suggested having the display consist of some number of zones that ı be independently powered up or down. In Fig. 5-50, we depict 16 zones using hed lines to separate them. When the cursor is in window 2, as shown in . 5-50(a), only the four zones in the lower righthand corner have to be lit up. : other 12 can be dark, saving 3/4 of the screen power.

When the user moves the cursor to window 1, the zones for window 2 can be kened and the zones behind window 1 can be turned on. However, because ıdow 1 straddles 9 zones, more power is needed. If the window manager can se of what is happening, it can automatically move window 1 to fit into four es, with a kind of snap-to-zone action, as shown in Fig. 5-50(b). To achieve reduction from 9/16 of full power to 4/16 of full power, the window manager to understand power management or be capable of accepting instructions from ıe other piece of the system that does. Even more sophisticated would be the ity to partially illuminate a window that was not completely full (e.g., a win- / containing short lines of text could be kept dark on the right hand side).

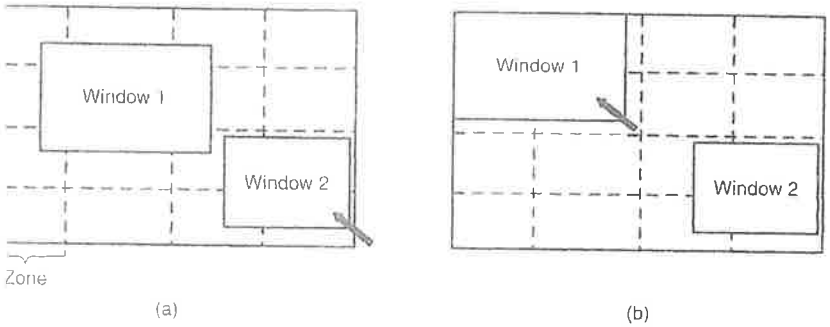


Figure 5-50. The use of zones for backlighting the display. (a) When window 2 is selected it is not moved. (b) When window 1 is selected, it moves to reduce the number of zones illuminated.

Hard Disk

Another major villain is the hard disk. It takes substantial energy to keep it ning at high speed, even if there are no accesses. Many computers, especially ps, spin the disk down after a certain number of minutes of activity. When it xt needed, it is spun up again. Unfortunately, a stopped disk is hibernating r than sleeping because it takes quite a few seconds to spin it up again, which :s noticeable delays for the user.

n addition, restarting the disk consumes considerable extra energy. As a :quence, every disk has a characteristic time, T_d, that is its break-even point, ...some time t in the future. If $t < T_d$, it takes less energy to keep the disk spinning rather than spin it down and then spin it up so quickly. If $t > T_d$, the energy saved makes it worth spinning the disk down and up again much later. If a good prediction could be made (e.g., based on past access patterns), the operating system could make good shutdown predictions and save energy. In practice, most systems are conservative and only spin down the disk after a few minutes of inactivity.

Another way to save disk energy is to have a substantial disk cache in RAM. If a needed block is in the cache, an idle disk does not have to be restarted to satisfy the read. Similarly, if a write to the disk can be buffered in the cache, a stopped disk does not have to restarted just to handle the write. The disk can remain off until the cache fills up or a read miss happens.

Another way to avoid unnecessary disk starts is for the operating system to keep running programs informed about the disk state by sending it messages or signals. Some programs have discretionary writes that can be skipped or delayed. For example, a word processor may be set up to write the file being edited to disk every few minutes. If the word processor knows that the disk is off at the moment it would normally write the file out, it can delay this write until the disk is next turned on or until a certain additional time has elapsed.

The CPU

The CPU can also be managed to save energy. A laptop CPU can be put to sleep in software, reducing power usage to almost zero. The only thing it can do in this state is wake up when an interrupt occurs. Therefore, whenever the CPU goes idle, either waiting for I/O or because there is no work to do, it goes to sleep.

On many computers, there is a relationship between CPU voltage, clock cycle, and power usage. The CPU voltage can often be reduced in software, which saves energy but also reduces the clock cycle (approximately linearly). Since power consumed is proportional to the square of the voltage, cutting the voltage in half makes the CPU about half as fast but at 1/4 the power.

This property can be exploited for programs with well-defined deadlines, such as multimedia viewers that have to decompress and display a frame every 40 msec, but go idle if they do it faster. Suppose that a CPU uses x joules while running full blast for 40 msec and $x/4$ joules running at half speed. If a multimedia viewer can decompress and display a frame in 20 msec, the operating system can run at full power for 20 msec and then shut down for 20 msec for a total energy usage of $x/2$ joules. Alternatively, it can run at half power and just make the deadline, but use only $x/4$ joules instead. A comparison of running at full speed and full power for some time interval and at half speed and one quarter power for twice as long is shown in Fig. 5-51. In both cases the same work is done, but in Fig. 5-51(b) only half the energy is consumed doing it.

threshold is about 120 dB, a dynamic range of a factor ...

confusion, A and B above are *amplitudes*. If we were to use the power level, ch is proportional to the square of the amplitude, the coefficient of the logam would be 10, not 20.

Audio waves can be converted to digital form by an **ADC (Analog Digital nverter)**. An ADC takes an electrical voltage as input and generates a binary nber as output. In Fig. 7-4(a) we see an example of a sine wave. To represent . signal digitally, we can sample it every ΔT seconds, as shown by the bar ghts in Fig. 7-4(b). If a sound wave is not a pure sine wave, but a linear superition of sine waves where the highest frequency component present is f, then it ufficient to make samples at a frequency $2f$. This result was proven mathematlly by H. Nyquist in 1924. Sampling more often is of no value since the higher quencies that such sampling could detect are not present.

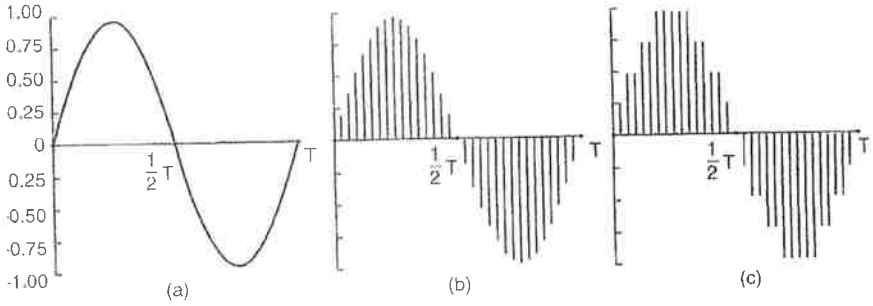


Figure 7-4. (a) A sine wave. (b) Sampling the sine wave. (c) Quantizing the samples to 4 bits.

Digital samples are never exact. The samples of Fig. 7-4(c) allow only nine lues, from −1.00 to +1.00 in steps of 0.25. Consequently, 4 bits are needed to present all of them. An 8-bit sample would allow 256 distinct values. A 16-bit mple would allow 65,536 distinct values. The error introduced by the finite mber of bits per sample is called the **quantization noise**. If it is too large, the r detects it.

Two well-known examples of sampled sound are the telephone and audio mpact discs. **Pulse code modulation** is used within the telephone system and es 7-bit (North America and Japan) or 8-bit (Europe) samples 8000 times per cond. This system gives a data rate of 56,000 bps or 64,000 bps. With only 00 samples/sec, frequencies above 4 kHz are lost.

Audio CDs are digital with a sampling rate of 44,100 samples/sec, enough to pture frequencies up to 22,050 Hz, which is good for people, bad for dogs. The mples are 16 bits each, and are linear over the range of amplitudes. Note that 6-bit samples allow only 65,536 distinct values, even though the dynamic range

Thus using only 16 bits per sample introduces some quantization noise (although the full dynamic range is not covered—CDs are not supposed to hurt). With 44,100 samples/sec of 16 bits each, an audio CD needs a bandwidth of 705.6 Kbps for monaural and 1.411 Mbps for stereo (see Fig. 7-2). Audio compression is possible based on psychoacoustic models of how human hearing works. A compression of 10x is possible using the MPEG layer 3 (MP3) system. Portable music players for this format have been common in recent years.

Digitized sound can easily be processed by computers in software. Dozens of programs exist for personal computers to allow users to record, display, edit, mix, and store sound waves from multiple sources. Virtually all professional sound recording and editing is digital nowadays.

7.2.2 Video Encoding

The human eye has the property that when an image is flashed on the retina, it is retained for some number of milliseconds before decaying. If a sequence of images is flashed at 50 or more images/sec, the eye does not notice that it is looking at discrete images. All video- and film-based motion picture systems exploit this principle to produce moving pictures.

To understand video systems, it is best to start with simple, old-fashioned black-and-white television. To represent the two-dimensional image in front of it as a one-dimensional voltage as a function of time, the camera scans an electron beam rapidly across the image and slowly down it, recording the light intensity as it goes. At the end of the scan, called a **frame**, the beam retraces. This intensity as a function of time is broadcast, and receivers repeat the scanning process to reconstruct the image. The scanning pattern used by both the camera and the receiver is shown in Fig. 7-5. (As an aside, CCD cameras integrate rather than scan, but some cameras and all CRT monitors do scan.)

The exact scanning parameters vary from country to country. NTSC has 525 scan lines, a horizontal to vertical aspect ratio of 4:3, and 30 frames/sec. The European PAL and SECAM systems have 625 scan lines, the same aspect ratio of 4:3, and 25 frames/sec. In both systems, the top few and bottom few lines are not displayed (to approximate a rectangular image on the original round CRTs). Only 483 of the 525 NTSC scan lines (and 576 of the 625 PAL/SECAM scan lines) are displayed.

While 25 frames/sec is enough to capture smooth motion, at that frame rate many people, especially older ones, will perceive the image to flicker (because the old image has faded off the retina before the new one appears). Rather than increase the frame rate, which would require using more scarce bandwidth, a different approach is taken. Instead of displaying the scan lines in order fromtop to bottom, first all the odd scan lines are displayed, then the even ones are displayed. Each of these half frames is called a **field**. Experiments have shown that although

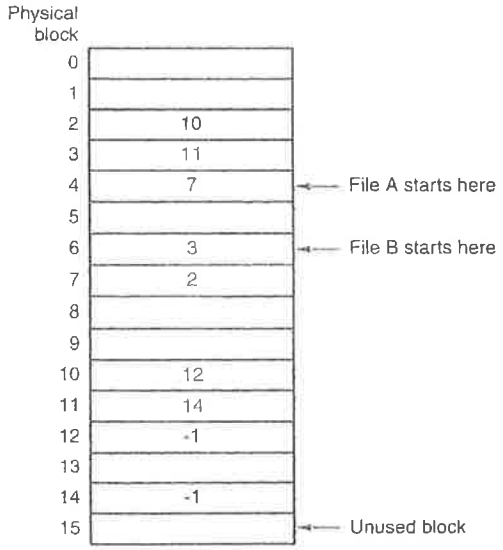


Figure 6-14. Linked list allocation using a file allocation table in main memory.

les

)ur last method for keeping track of which blocks belong to which file is to ·iate with each file a data structure called an **i-node (index-node)**, which lists ttributes and disk addresses of the file's blocks. A simple example is dep- in Fig. 6-15. Given the i-node, it is then possible to find all the blocks of the The big advantage of this scheme over linked files using an in-memory table ıt the i-node need only be in memory when the corresponding file is open. If i-node occupies n bytes and a maximum of k files may be open at once, the memory occupied by the array holding the i-nodes for the open files is only ·tes. Only this much space need be reserved in advance.

'his array is usually far smaller than the space occupied by the file table ·ibed in the previous section. The reason is simple. The table for holding the d list of all disk blocks is proportional in size to the disk itself. If the disk ı blocks, the table needs n entries. As disks grow larger, this table grows rly with them. In contrast, the i-node scheme requires an array in memory e size is proportional to the maximum number of files that may be open at It does not matter if the disk is 1 GB or 10 GB or 100 GB.

)ne problem with i-nodes is that if each one has room for a fixed number of addresses, what happens when a file grows beyond this limit? One solution reserve the last disk address not for a data block, but instead for the address

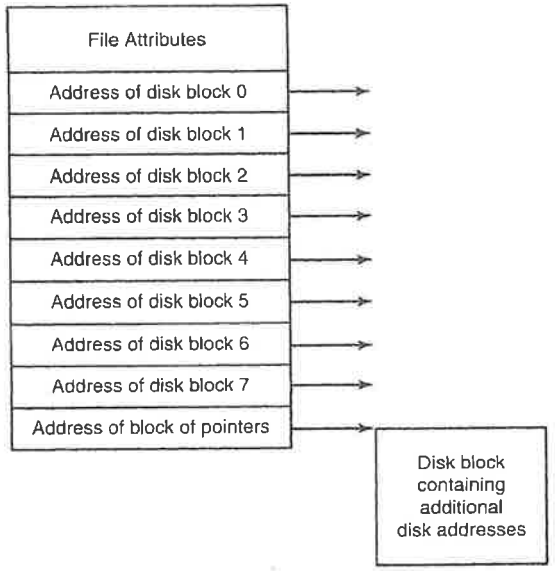


Figure 6-15. An example i-node.

of a block containing more disk block addresses, as shown in Fig. 6-15. Even more advanced would be two or more such blocks containing disk addresses or even disk blocks pointing to other disk blocks full of addresses. We will come back to i-nodes when studying UNIX later.

6.3.3 Implementing Directories

Before a file can be read, it must be opened. When a file is opened, the operating system uses the path name supplied by the user to locate the directory entry. The directory entry provides the information needed to find the disk blocks. Depending on the system, this information may be the disk address of the entire file (contiguous allocation), the number of the first block (both linked list schemes), or the number of the i-node. In all cases, the main function of the directory system is to map the ASCII name of the file onto the information needed to locate the data.

A closely related issue is where the attributes should be stored. Every file system maintains file attributes, such as each file's owner and creation time, and they must be stored somewhere. One obvious possibility is to store them directly in the directory entry. Many systems do precisely that. This option is shown in Fig. 6-16(a) In this simple design, a directory consists of a list of fixed-size