**INPUT/OUTPUT**

****

**by**

**FIRDA IMAM ASA (L200164001)**

**BERLIAN EDRA BAGASKARA (L200164013)**

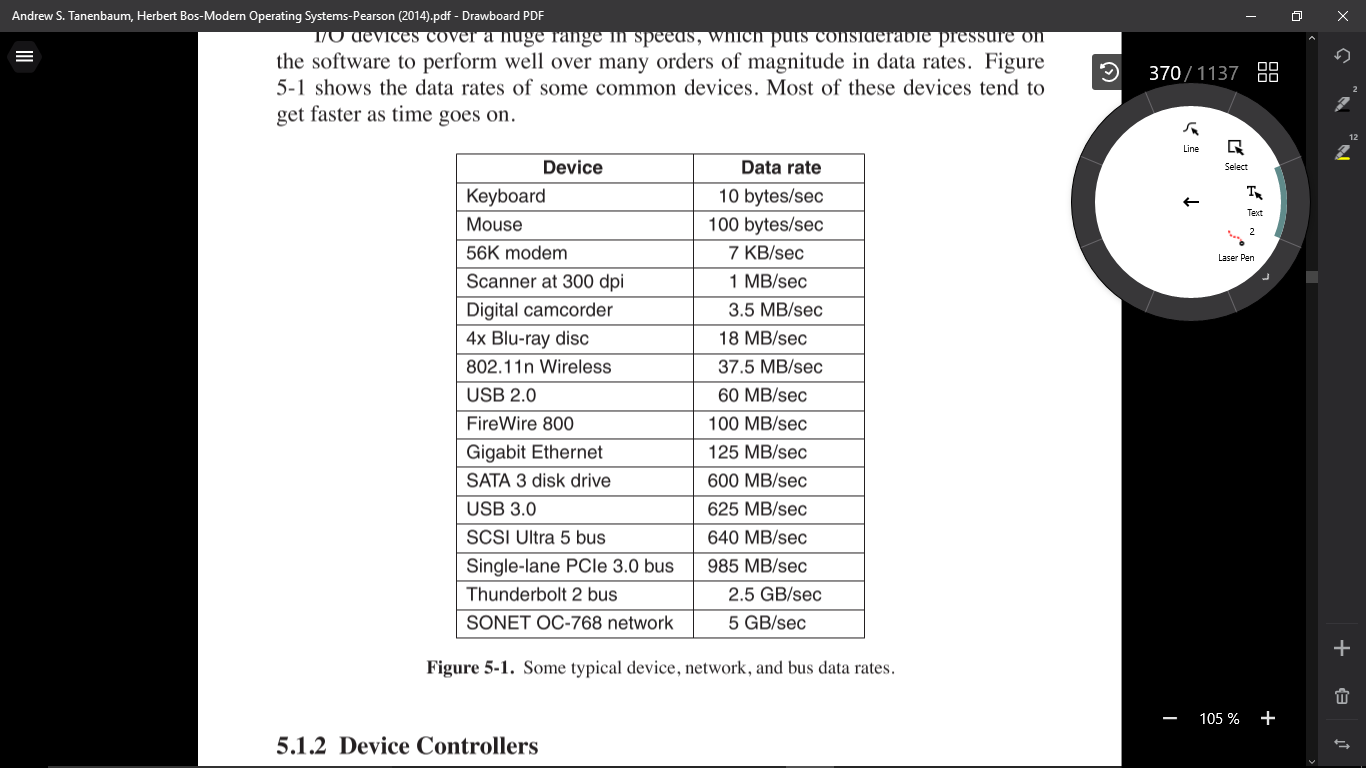
**X Class**

In addition to providing abstractions such as processes, address spaces, and files, an operating system also controls all the computer’s I/O (Input/Output) de-vices.

1. **PRINCIPLES OF I/O HARDWARE**
2. **I/O Devices**

I/O devices can be roughly divided into two categories: block devices and character devices. I/O devices cover a huge range in speeds, which puts considerable pressure on the software to perform well over many orders of magnitude in data rates. Figure 5-1 shows the data rates of some common devices. Most of these devices tend to

get faster as time goes on.



1. **Device Controllers**

I/O units often consist of a mechanical component and an electronic component. It is possible to separate the two portions to provide a more modular and general design. The electronic component is called the device controller or adapter. On personal computers, it often takes the form of a chip on the par-

entboard or a printed circuit card that can be inserted into a (PCIe) expansion slot.

1. **Memory-Mapped I/O**

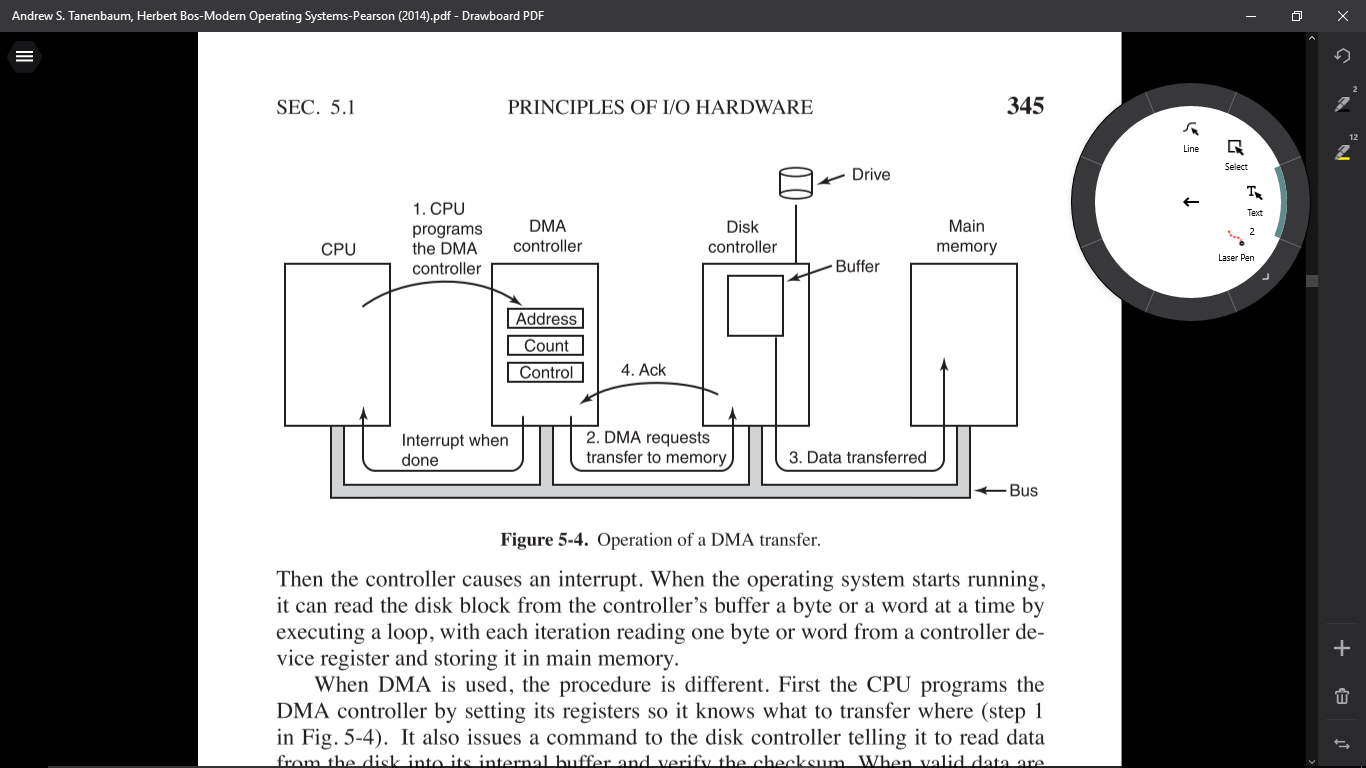
Each controller has a few registers that are used for communicating with the CPU. By writing into these registers, the operating system can command the device to deliver data, accept data, switch itself on or off, or otherwise perform some action. By reading from these registers, the operating system can learn what the

device’s state is, whether it is prepared to accept a new command, and so on.

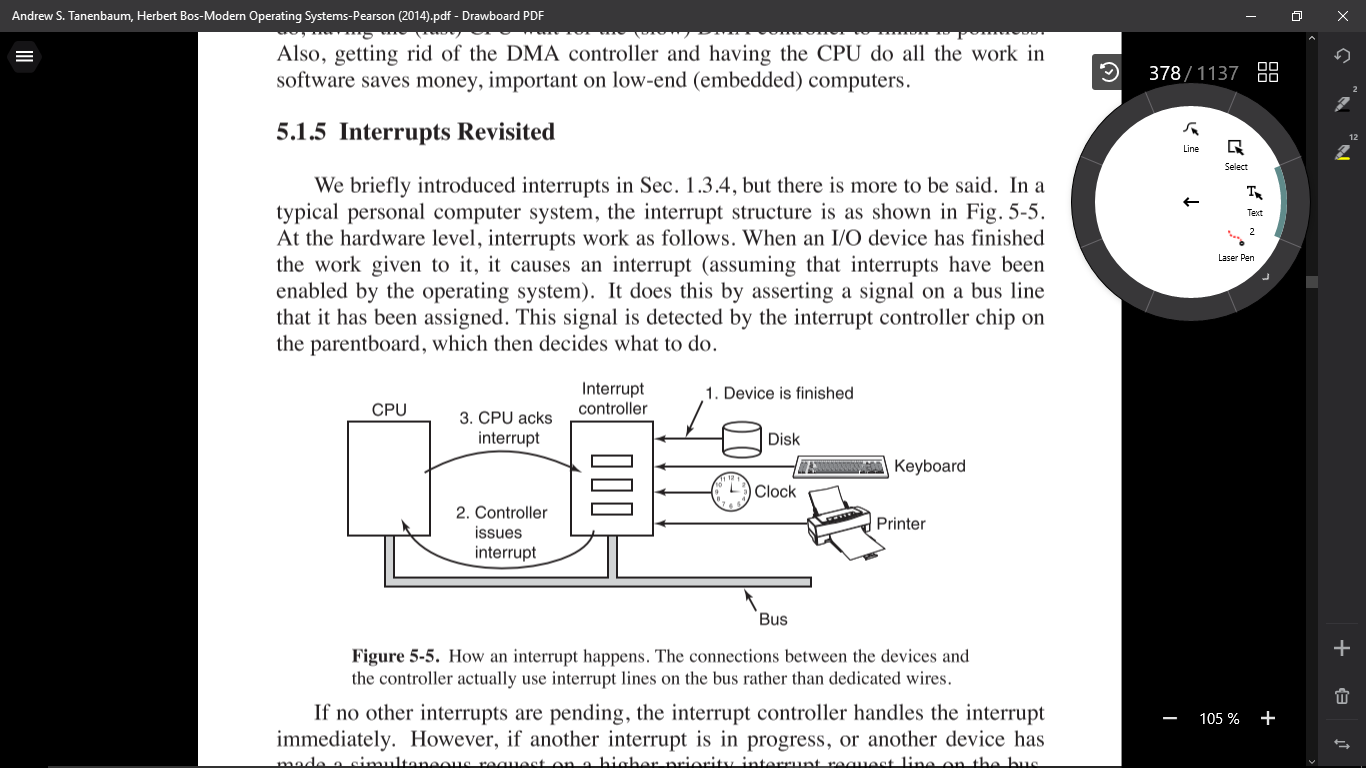
1. **Direct Memory Access**

No matter whether a CPU does or does not have memory-mapped I/O, it needs to address the device controllers to exchange data with them. The CPU can request data from an I/O controller one byte at a time, but doing so wastes the CPU’s time,

so a different scheme, called DMA (Direct Memory Access) is often used.



1. **Interrupts Revisited**

When an I/O device has finished the work given to it, it causes an interrupt (assuming that interrupts have been enabled by the operating system). It does this by asserting a signal on a bus line that it has been assigned. This signal is detected by the interrupt controller chip on the parentboard, which then decides what to do.

1. **PRINCIPLES OF I/O SOFTWARE**
2. **Goals of the I/O Software**

A key concept in the design of I/O software is known as device independence. What it means is that we should be able to write programs that can access any I/O device without having to specify the device in advance. For example, a program that reads a file as input should be able to read a file on a hard disk, a DVD, or on a

USB stick without having to be modified for each different device.

1. **Programmed I/O**

There are three fundamentally different ways that I/O can be performed. In this section we will look at the first one (programmed I/O). In the next two sections we will examine the others (interrupt-driven I/O and I/O using DMA). The simplest form of I/O is to have the CPU do all the work. This method is called pro-

grammed I/O.

1. **Interrupt-Driven I/O**

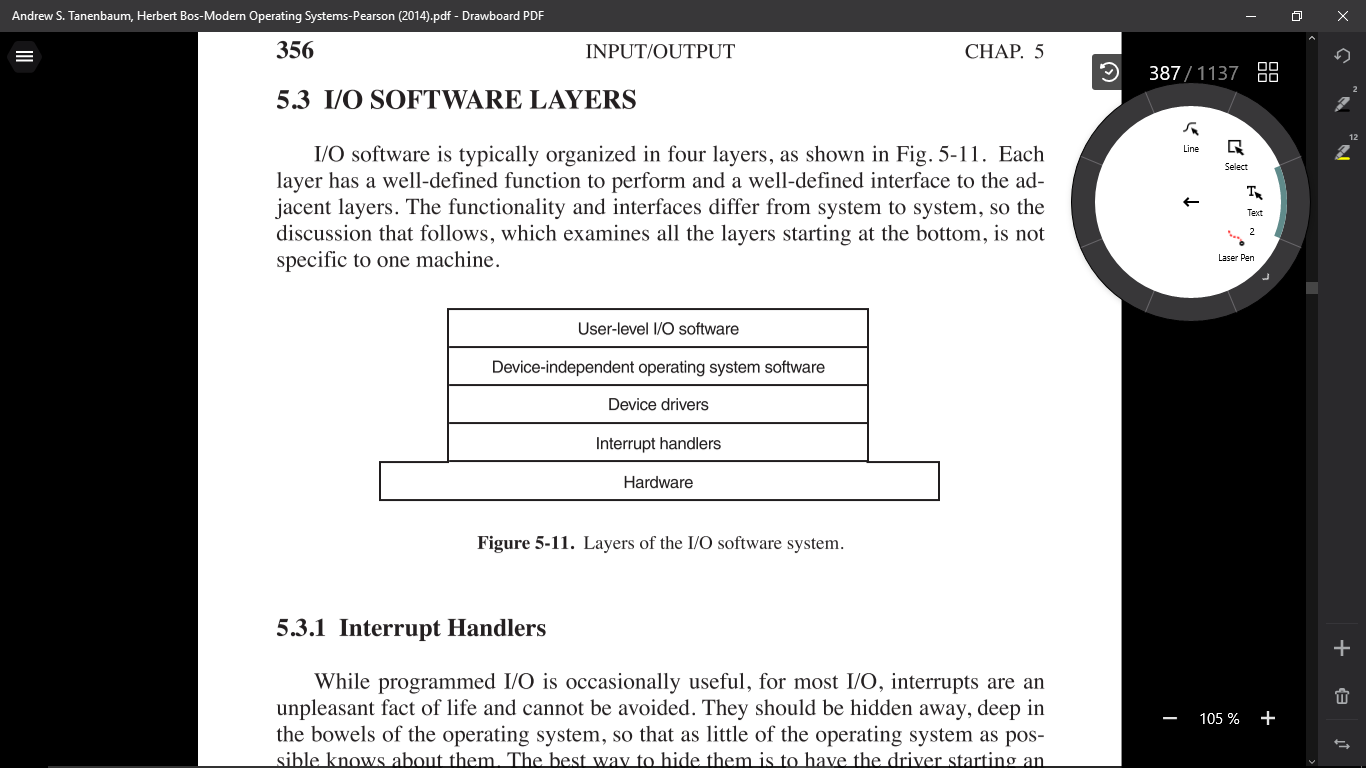
Now let us consider the case of printing on a printer that does not buffer characters but prints each one as it arrives. If the printer can print, say 100 characters/sec, each character takes 10 msec to print. This means that after every character is written to the printer’s data register, the CPU will sit in an idle loop for 10 msec waiting to be allowed to output the next character. This is more than enough time to do a context switch and run some other process for the 10 msec that would otherwise be wasted. The way to allow the CPU to do something else while waiting for the printer to become ready is to use interrupts. When the system call to print the string is made, the buffer is copied to kernel space, as we showed earlier, and the first character is copied to the printer as soon as it is willing to accept a character. At that point the CPU calls the scheduler and some other process is run. The process that asked for the string to be printed is blocked until the entire string has printed. The work done

on the system call

1. **I/O Using DMA**

An obvious disadvantage of interrupt-driven I/O is that an interrupt occurs on ev ery character. Interrupts take time, so this scheme wastes a certain amount of CPU time. A solution is to use DMA. Here the idea is to let the DMA controller feed the characters to the printer one at time, without the CPU being bothered. In essence, DMA is programmed I/O, only with the DMA controller doing all the work, instead of the main CPU. This strategy requires special hardware (the DMA

controller) but frees up the CPU during the I/O to do other work.

1. **I/O SOFTWARE LAYERS**
2. **DISKS**

* **Disk Hardware**

Disks come in a variety of types. The most common ones are the magnetic hard disks. They are characterized by the fact that reads and writes are equally fast, which makes them suitable as secondary memory (paging, file systems, etc.). Arrays of these disks are sometimes used to provide highly reliable storage.

* **Disk Formatting**

A hard disk consists of a stack of aluminum, alloy, or glass platters typically 3.5 inch in diameter (or 2.5 inch on notebook computers). On each platter is deposited a thin magnetizable metal oxide. After manufacturing, there is no information whatsoever on the disk.

* **Disk ArmScheduling Algorithm**

In this section we will look at some issues related to disk drivers in general. First, consider how long it takes to read or write a disk block. The time required is determined by three factors:

1. Seek time (the time to move the arm to the proper cylinder).
2. Rotational delay (how long for the proper sector to appear under the reading head).
3. Actual data transfer time

For most disks, the seek time dominates the other two times, so reducing the mean seek time can improve system performance substantially.

* **Error Handling**

Disk manufacturers are constantly pushing the limits of the technology by increasing linear bit densities. A track midway out on a 5.25-inch disk has a circumference of about 300 mm. If the track holds 300 sectors of 512 bytes, the linear recording density may be about 5000 bits/mm taking into account the fact that some space is lost to preambles, ECCs, and intersector gaps. Recording 5000 bits/mm requires an extremely uniform substrate and a very fine oxide coating. Unfortunately, it is not possible to manufacture a disk to such specifications without defects. As soon as manufacturing technology has improved to the point where it is possible to operate flawlessly at such densities, disk designers will go to higher densities to increase the capacity. Doing so will probably reintroduce defects.

* **Stable Storage**

As we have seen, disks sometimes make errors. Good sectors can suddenly become bad sectors. Whole drives can die unexpectedly. RAIDs protect against a few sectors going bad or even a drive falling out. However, they do not protect against write errors laying down bad data in the first place. They also do not protect against crashes during writes corrupting the original data without replacing them by newer data.

1. **Clocks**

Clocks (also called timers) are essential to the operation of any multiprogrammed system for a variety of reasons. They maintain the time of day and prevent one process from monopolizing the CPU, among other things. The clock software can take the form of a device driver, even though a clock is neither a block device, like a disk, nor a character device, like a mouse. Our examination of clocks will follow the same pattern as in the previous section: first a look at clock hardware and then a look at the clock software.

* **Clock Hardware**

Tw o types of clocks are commonly used in computers, and both are quite different from the clocks and watches used by people. The simpler clocks are tied to the 110- or 220-volt power line and cause an interrupt on every voltage cycle, at 50 or 60 Hz. These clocks used to dominate, but are rare nowadays.

* **Clock Software**

All the clock hardware does is generate interrupts at known intervals. Everything else involving time must be done by the software, the clock driver. The exact duties of the clock driver vary among operating systems, but usually include most of the following:

1. Maintaining the time of day.
2. Preventing processes from running longer than they are allowed to.
3. Accounting for CPU usage.
4. Handling the alar m system call made by user processes.
5. Providing watchdog timers for parts of the system itself.
6. Doing profiling, monitoring, and statistics gathering.

* **Soft Timers**

Most computers have a second programmable clock that can be set to cause timer interrupts at whatever rate a program needs. This timer is in addition to the main system timer whose functions were described above. As long as the interrupt frequency is low, there is no problem using this second timer for application-specific purposes. The trouble arrives when the frequency of the application-specific

SEC. 5.5 CLOCKS 393

timer is very high. Below we will briefly describe a software-based timer scheme that works well under many circumstances, even at fairly high frequencies. The idea is due to Aron and Druschel (1999). For more details, please see their paper.

1. **User Interfaces: Keyboard, Mouse, Monitor**

* **Input Software**

1. **Keyboard Software**

The number in the I/O register is the key number, called the scan code, not the ASCII code. Normal keyboards have fewer than 128 keys, so only 7 bits are needed to represent the key number. The eighth bit is set to 0 on a key press and to 1 on a key release. It is up to the driver to keep track of the status of each key (up or down). So all the hardware does is give press and release interrupts. Software does the rest.

1. **Mouse Software**

Most PCs have a mouse, or sometimes a trackball, which is just a mouse lying on its back. One common type of mouse has a rubber ball inside that protrudes through a hole in the bottom and rotates as the mouse is moved over a rough surface. As the ball rotates, it rubs against rubber rollers placed on orthogonal shafts. Motion in the east-west direction causes the shaft parallel to the y-axis to rotate; motion in the north-south direction causes the shaft parallel to the x-axis to rotate.

* **Output Software**

1. **Text Windows**

Output is simpler than input when the output is sequentially in a single font, size, and color. For the most part, the program sends characters to the current window and they are displayed there. Usually, a block of characters, for example, a line, is written in one system call.

1. **The X Windows System**

Nearly all UNIX systems base their user interface on the X Window System (often just called X), developed at M.I.T. as part of project Athena in the 1980s. It is very portable and runs entirely in user space. It was originally intended for connecting a large number of remote user terminals with a central compute server, so it is logically split into client software and host software, which can potentially run on different computers. On modern personal computers, both parts can run on the same machine. On Linux systems, the popular Gnome and KDE desktop environments run on top of X.

1. **Graphical User Interfaces**

Most personal computers offer a GUI (Graphical User Interface). The acronym GUI is pronounced ‘‘gooey.’’ The GUI was invented by Douglas Engelbart and his research group at the Stanford Research Institute. It was then copied by researchers at Xerox PARC. One fine day, Steve Jobs, cofounder of Apple, was touring PARC and saw a GUI on a Xerox computer and said something to the effect of ‘‘Holy mackerel. This is the future of computing.’’ The GUI gav e him the idea for a new computer, which became the Apple Lisa. The Lisa was too expensive and was a commercial failure, but its successor, the Macintosh, was a huge success.

1. **Fonts**

In versions of Windows before 3.1, characters were represented as bitmaps and copied onto the screen or printer using BitBlt. The problem with that, as we just saw, is that a bitmap that makes sense on the screen is too small for the printer. Also, a different bitmap is needed for each character in each size. In other words, given the bitmap for A in 10-point type, there is no way to compute it for 12-point type. Because every character of every font might be needed for sizes ranging from 4 point to 120 point, a vast number of bitmaps were needed. The whole system was just too cumbersome for text.

1. **Touch Screen**

More and more the screen is used as an input device also. Especially on smartphones, tablets and other ultra-portable devices it is convenient to tap and swipe aw ay at the screen with your finger (or a stylus). The user experience is different and more intuitive than with a mouse-like device, since the user interacts directly with the objects on the screen. Research has shown that even orangutans and other primates like little children are capable of operating touch-based devices.

1. **Thin Clients**

Over the years, the main computing paradigm has oscillated between centralized and decentralized computing. The first computers, such as the ENIAC, were, in fact, personal computers, albeit large ones, because only one person could use one at once. Then came timesharing systems, in which many remote users at simple terminals shared a big central computer. Next came the PC era, in which the users had their own personal computers again. While the decentralized PC model has advantages, it also has some severe disadvantages that are only beginning to be taken seriously. Probably the biggest problem is that each PC has a large hard disk and complex software that must be maintained. For example, when a new release of the operating system comes out, a great deal of work has to be done to perform the upgrade on each machine separately. At most corporations, the labor costs of doing this kind of software maintenance dwarf the actual hardware and software costs. For home users, the labor is technically free, but few people are capable of doing it correctly and fewer still enjoy doing it. With a centralized system, only one or a few machines have to be updated and those machines have a staff of experts to do the work.

1. **Power Management**

The first general-purpose electronic computer, the ENIAC, had 18,000 vacuum tubes and consumed 140,000 watts of power. As a result, it ran up a nontrivial electricity bill. After the invention of the transistor, power usage dropped dramatically and the computer industry lost interest in power requirements. However, nowadays power management is back in the spotlight for several reasons, and the operating system is playing a role here.

* **Hardware Issues**

Batteries come in two general types: disposable and rechargeable. Disposable batteries (most commonly AAA, AA, and D cells) can be used to run handheld devices, but do not have enough energy to power notebook computers with large bright screens. A rechargeable battery, in contrast, can store enough energy to power a notebook for a few hours. Nickel cadmium batteries used to dominate here, but they gav e way to nickel metal hydride batteries, which last longer and do not pollute the environment quite as badly when they are eventually discarded. Lithium ion batteries are even better, and may be recharged without first being fully drained, but their capacities are also severely limited

* **Operating System Issues**

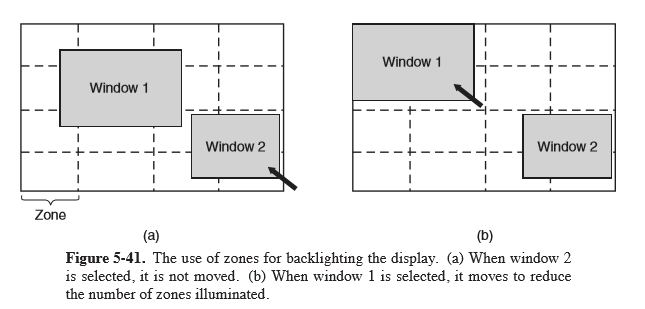
The operating system plays a key role in energy management. It controls all the devices, so it must decide what to shut down and when to shut it down. If it shuts down a device and that device is needed again quickly, there may be an annoying delay while it is restarted. On the other hand, if it waits too long to shut down a device, energy is wasted for nothing.

* **The Display**

Let us now look at the big spenders of the energy budget to see what can be done about each one. One of the biggest items in everyone’s energy budget is the display. To get a bright sharp image, the screen must be backlit and that takes substantial energy. Many operating systems attempt to save energy here by shutting down the display when there has been no activity for some number of minutes. Often the user can decide what the shutdown interval is, thus pushing the trade-off between frequent blanking of the screen and draining the battery quickly back to the user (who probably really does not want it). Turning off the display is a sleep state because it can be regenerated (from the video RAM) almost instantaneously when any key is struck or the pointing device is moved.

* **The Hard Disk**

Another major villain is the hard disk. It takes substantial energy to keep it spinning at high speed, even if there are no accesses. Many computers, especially notebooks, spin the disk down after a certain number of minutes of being idle.



When it is next needed, it is spun up again. Unfortunately, a stopped disk is hibernating rather than sleeping because it takes quite a few seconds to spin it up again, which causes noticeable delays for the user.

* **The CPU**

The CPU can also be managed to save energy. A notebook 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.

* **The Memory**

Tw o possible options exist for saving energy with the memory. First, the cache can be flushed and then switched off. It can always be reloaded from main memory with no loss of information. The reload can be done dynamically and quickly, so turning off the cache is entering a sleep state.

* **Wireless Communication**

Increasingly many portable computers have a wireless connection to the outside world (e.g., the Internet). The radio transmitter and receiver required are often first-class power hogs. In particular, if the radio receiver is always on in order to listen for incoming email, the battery may drain fairly quickly. On the other hand, if the radio is switched off after, say, 1 minute of being idle, incoming messages may be missed, which is clearly undesirable.

* **Thermal Management**

A somewhat different, but still energy-related issue, is thermal management. Modern CPUs get extremely hot due to their high speed. Desktop machines normally have an internal electric fan to blow the hot air out of the chassis. Since reducing power consumption is usually not a driving issue with desktop machines, the fan is usually on all the time.

* **Battery Management**

In ye olde days, a battery just provided current until it was fully drained, at which time it stopped. Not any more. Mobile devices now use smart batteries now, which can communicate with the operating system.

* **Driver Interface**

Several operating systems have an elaborate mechanism for doing power management called ACPI (Advanced Configuration and Power Interface). The operating system can send any conformant driver commands asking it to report on the capabilities of its devices and their current states. This feature is especially important when combined with plug and play because just after it is booted, the operating system does not even know what devices are present, let alone their properties with respect to energy consumption or power manageability.

1. **Research On Input / Output**

There is a fair amount of research on input/output. Some of it is focused on specific devices, rather than I/O in general. Other work focuses on the entire I/O infrastructure. For instance, the Streamline architecture aims to provide application-tailored I/O that minimizes overhead due to copying, context switching, signaling and poor use of the cache and TLB (DeBruijn et al., 2011). It builds on the notion of Beltway Buffers, advanced circular buffers that are more efficient than existing buffering systems (DeBruijn and Bos, 2008). Streamline is especially useful for demanding network applications. Megapipe (Han et al., 2012) is another network I/O architecture for message-oriented workloads. It creates per-core bidirectional channels between the kernel and user space, on which the systems layers abstractions like lightweight sockets. The sockets are not quite POSIX-compliant, so applications need to be adapted to benefit from the more efficient I/O.

Often, the goal of the research is to improve performance of a specific device in one way or another. Disk systems are a case in point. Disk-arm scheduling algorithms are an ever-popular research area. Sometimes the focus is on improved peformance (Gonzalez-Ferez et al., 2012; Prabhakar et al., 2013; and Zhang et al., 2012b) but sometimes it is on lower energy usage (Krish et al., 2013; Nijim et al., 2013; and Zhang et al., 2012a). With the popularity of server consolidation using virtual machines, disk scheduling for virtualized systems has become a hot topic (Jin et al., 2013; and Ling et al., 2012).