

I/O Devices

Before delving into the main content of this part of the book (on persistence), we first introduce the concept of an **input/output (I/O) device** and show how the operating system might interact with such an entity. I/O is quite critical to computer systems, of course; imagine a program without any input (it produces the same result each time); now imagine a program with no output (what was the purpose of it running?). Clearly, for computer systems to be interesting, both input and output are required. And thus, our general problem:

CRUX: HOW TO INTEGRATE I/O INTO SYSTEMS

How should I/O be integrated into systems? What are the general mechanisms? How can we make them efficient?

36.1 System Architecture

To begin our discussion, let's look at the structure of a typical system (Figure 36.1). The picture shows a single CPU attached to the main memory of the system via some kind of **memory bus** or interconnect. Some devices are connected to the system via a general **I/O bus**, which in many modern systems would be **PCI** (or one of its many derivatives); graphics and some other higher-performance I/O devices might be found here. Finally, even lower down are one or more of what we call a **peripheral bus**, such as **SCSI**, **SATA**, or **USB**. These connect the slowest devices to the system, including **disks**, **mice**, and other similar components.

One question you might ask is: why do we need a hierarchical structure like this? Put simply: physics, and cost. The faster a bus is, the shorter it must be; thus, a high-performance memory bus does not have much room to plug devices and such into it. In addition, engineering a bus for high performance is quite costly. Thus, system designers have adopted this hierarchical approach, where components that demand high performance (such as the graphics card) are nearer the CPU. Lower per-

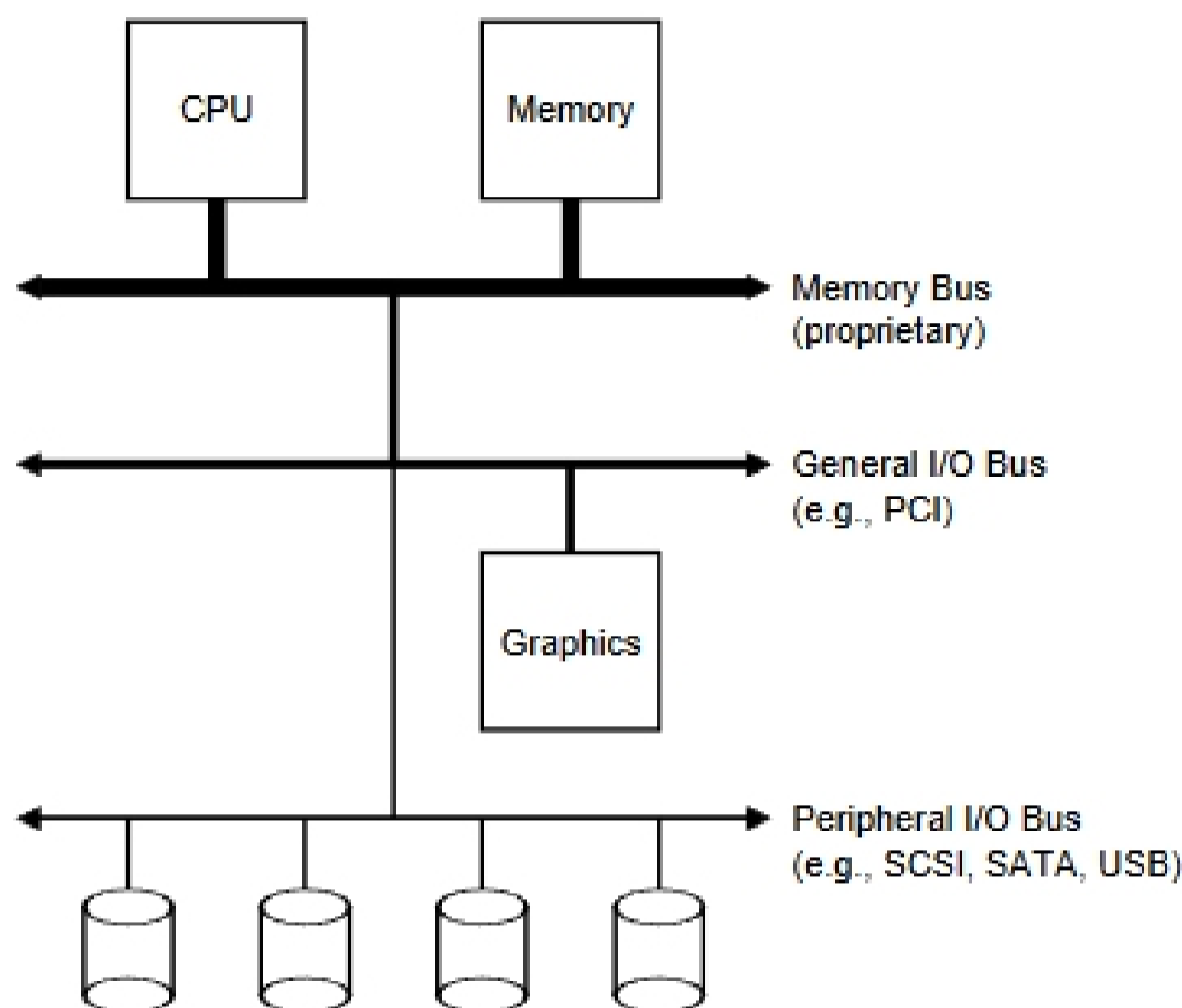


Figure 36.1: Prototypical System Architecture

formance components are further away. The benefits of placing disks and other slow devices on a peripheral bus are manifold; in particular, you can place a large number of devices on it.

36.2 A Canonical Device

Let us now look at a canonical device (not a real one), and use this device to drive our understanding of some of the machinery required to make device interaction efficient. From Figure 36.2, we can see that a device has two important components. The first is the hardware **interface** it presents to the rest of the system. Just like a piece of software, hardware must also present some kind of interface that allows the system software to control its operation. Thus, all devices have some specified interface and protocol for typical interaction.

The second part of any device is its **internal structure**. This part of the device is implementation specific and is responsible for implementing the abstraction the device presents to the system. Very simple devices will have one or a few hardware chips to implement their functionality; more complex devices will include a simple CPU, some general purpose memory, and other device-specific chips to get their job done. For example, modern RAID controllers might consist of hundreds of thousands of lines of **firmware** (i.e., software within a hardware device) to implement its functionality.



Figure 36.2: A Canonical Device

36.3 The Canonical Protocol

In the picture above, the (simplified) device interface is comprised of three registers: a **status** register, which can be read to see the current status of the device; a **command** register, to tell the device to perform a certain task; and a **data** register to pass data to the device, or get data from the device. By reading and writing these registers, the operating system can control device behavior.

Let us now describe a typical interaction that the OS might have with the device in order to get the device to do something on its behalf. The protocol is as follows: