Lecture 6 — Input/Output Devices & Drivers

Jeff Zarnett
jzarnett@uwaterloo.ca

Department of Electrical and Computer Engineering
University of Waterloo

April 8, 2024

The Importance of I/O

Though the computer's name and typical use suggests the purpose of the machine is computation, input/output is just as important to the usefulness.

A computer that takes no inputs and produces no outputs is not very useful.

Schrödinger's computer.

I/O is Messy

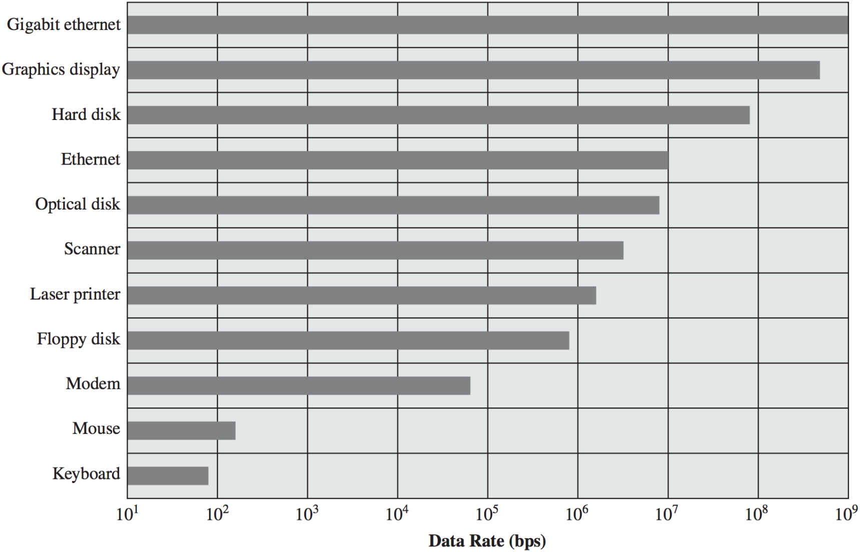
I/O is, unfortunately, a messy business.

There are only a small number of CPU types your typical desktop OS might run (AMD and Intel processors, for example, can execute the same binary code).

There are uncountably many I/O devices in the world. And they're all different.

Keyboards, hard drives, printers, and headsets are all I/O devices, but they serve very different purposes and work in different ways.

I/O Device Rates



I/O is Messy

We might like to think that USB has taken away some of the complexity, but that's just the way a device connects to the computer.

Managing the device itself is still as complicated as ever.

All of the examples just listed can be connected via USB (Universal Serial Bus).

Disk vs General I/O

Disk I/O has a huge impact on performance; it will receive its own discussion.

But first, a little discussion about I/O in general.

Disk vs General I/O

For I/O the key parts are the bus and a controller.

After that there will be a protocol for how the devices will communicate e.g., polling, interrupts, or DMA.

Communication Protocol

Our options: polling, interrupts, and DMA.
Not all are equally good options in efficiency terms.

An I/O port represents the connection and may have 4 registers:
data-in, data-out, status, control.

Main system is called host.

Polling

Last choice, but might be our only choice!

Use a bit to mark the device as busy during operation.

Not exciting, but it does work.

Polling

Periodic polling might decrease CPU cycles but...

"DO IT?"
DAN, I'M NOT A REPUBLIC SERIAL VILLAIN. DO YOU SERIOUSLY THINK I'D EXPLAIN MY MASTER-STROKE IF THERE REMAINED THE SLIGHTEST CHANCE OF YOU AFFECTING ITS OUTCOME?
I DID IT THIRTY-FIVE MINUTES AGO.

Interrupts

There’s some sort of interrupt request line (hardware) and when the CPU sees the signal that indicates the presence of an interrupt.

Then the interrupt handler is executed to deal with it and clear the signal.

We just learned about the implementation of this at the lower levels.

Something Something Fire Drill

Ignores some complexity: interrupt priority, disabling interrupts?

But can avoid the problem of missing data.

DMA

Based on the idea of delegating the work to the DMA controller.

Can be quite efficient: CPU is still interrupted, but way less.

Application I/O Interface

Ideally a general-purpose operating system will accept new devices being added to the system without editing/reinstalling/recompiling the code.

Your experience with object oriented programming gives you some familiarity with the solution of how we should accomplish this goal.

We want to abstract away the details of the hardware, to the extent we can.

Provide a uniform interface to interact with.

I/O Interfaces

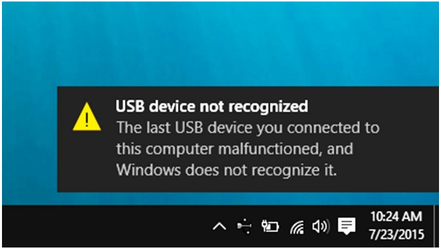
In the very early days of operating systems, the hardware the computer shipped with was all the hardware it ever supported.

If the vendor came out with a new module, they would have a new operating system update to introduce support for that device.

I/O Interfaces

This got to be unmanageable in the era of the IBM PC because anybody could create hardware and attach it via a standard interface.

Relying on IBM or Microsoft or whoever your OS vendor was to implement support for a random piece of hardware was not realistic.

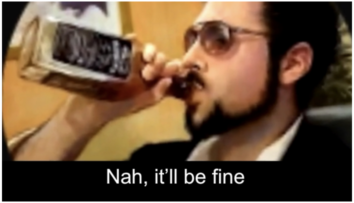


USB device not recognized
The last USB device you connected to this computer malfunctioned, and Windows does not recognize it.

I/O Interfaces

Operating system developers thought they were very clever.

They realized that they could shift the work to the hardware developers through a concept called device drivers.



I/O Interfaces

The device driver plugs in to the operating system through a standard interface.

It tells the operating system a bit about the hardware and translates commands from the operating system to hardware instructions.

Hardware developers often made extremely poor drivers.

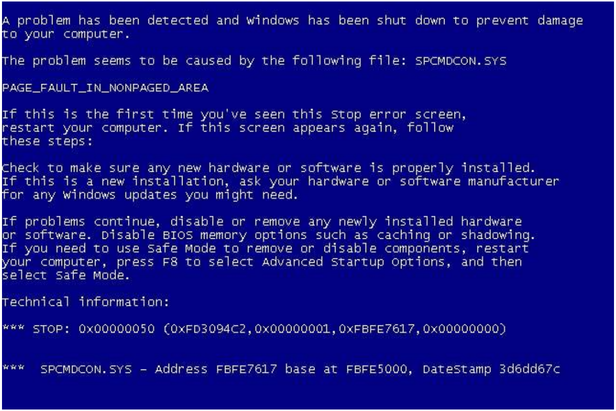
Device Drivers

The problem was exacerbated by a Windows design decision.

Device drivers run in the system at the same protection level as the kernel.

Some other operating systems have user-mode drivers, where possible or at an intermediate level between that of user space and the kernel.

Device Drivers



```
A problem has been detected and Windows has been shut down to prevent damage
to your computer.

The problem seems to be caused by the following file: SPCMDCON.SYS

PAGE_FAULT_IN_NONPAGED_AREA

If this is the first time you've seen this Stop error screen,
restart your computer. If this screen appears again, follow
these steps:

Check to make sure any new hardware or software is properly installed.
If this is a new installation, ask your hardware or software manufacturer
for any Windows updates you might need.

If problems continue, disable or remove any newly installed hardware
or software. Disable BIOS memory options such as caching or shadowing.
If you need to use Safe Mode to remove or disable components, restart
your computer, press F8 to select Advanced Startup Options, and then
select Safe Mode.

Technical information:

*** STOP: 0x00000050 (0xFD3094C2,0x00000001,0xFBFE7617,0x00000000)


***  SPCMDCON.SYS - Address FBFE7617 base at FBFE5000, DateStamp 3d6dd67c
```

In Windows, a driver can invoke a system call that brings up everyone's favourite feature: the Blue Screen of Death (BSOD).

Blue Screen of Death

Microsoft, rightly or wrongly, was blamed for a lot of those BSODs.

They took two approaches to remedy this problem.

One was to write and include in Windows a lot more device drivers.

Blue Screen of Death

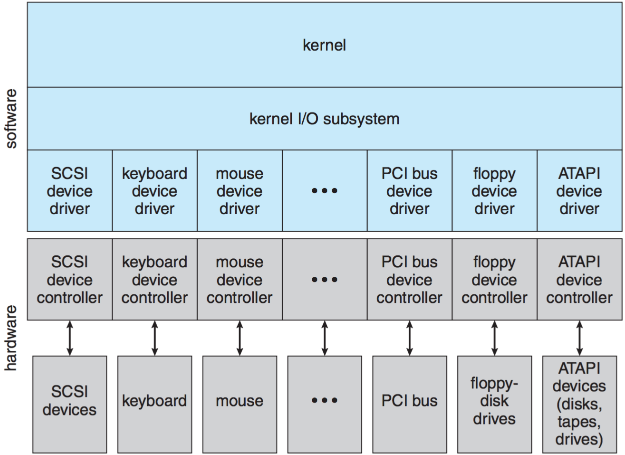
The other was to introduce the static driver verifier; software used to test, at compile/build time, whether the driver will behave badly.

Passing this test is required to get a sticker of approval from Microsoft.

The battle rages on about who is responsible for writing the drivers.

Device Drivers

Device drivers connect into the kernel’s I/O subsystem to mediate between the kernel’s I/O subsystem and the hardware device controller.



Abstraction

Abstracting away details of the hardware makes the job of the OS dev easier.

Part of the difficulty is that devices can vary on numerous dimensions:

- **Data transfer mode**
- **Access method**
- **Transfer schedule**
- **Dedication**
- **Device Speed**
- **Transfer Direction**

Abstraction

We would like to, as much as possible, keep the details above from the OS.

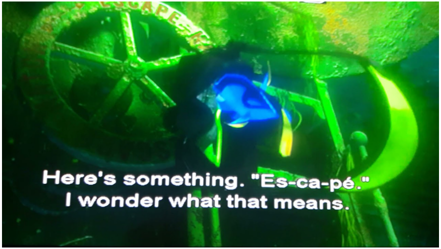
Devices will typically be grouped into a few categories so appropriate system calls can be issued.

If a device is block-oriented, the OS should be issuing block read and write commands, not trying to do it one character at a time.

Escape

Operating systems also usually have an escape system call that allows passing of a command directly from an application or the kernel to a device driver.

This allows us to issue commands to a device that the OS designers have not thought of and created system calls for.



Escape

The UNIX system call for this is `ioctl` (“I/O Control”).

It takes three parameters:

- A file descriptor indicating the hardware device.
- The command number
- A pointer to an arbitrary data structure that has control info & data.

Block and Character I/O

The block device interface is used for block devices such as hard disk drives.

Any device will support `read` and `write` commands, and if it is a random-access device, it will have a `seek` command to jump to a specific block.

An application usually accesses the hard disk through the file system.

The OS can work on the hard drive using these 2/3 commands without being concerned with how that command is actually transmitted.

Block and Character I/O

We can abstract things a little bit further, from the perspective of the application developer, by having a memory-mapped file.

Then, rather than using the block oriented operations directly, the application just writes to and reads from “memory”.

The OS handles the behind-the-scene coordination to make writes go out to the correct block and read from the correct block.

Block and Character I/O

A character-oriented device is something like the keyboard.
The system calls are `get` and `put`.

Libraries and other structures may exist to work on a whole line at a time.

This is a good match for input devices that produce data in small amounts and at unpredictable times.

Perhaps also for printers and sound output which operate naturally on a linear stream of bytes.

Network Devices

Network devices are fundamentally different from those that are directly attached to the system.

Thus, the `read`, `write`, and `seek` routines are not really appropriate.

The model in UNIX and Windows is that of sockets.

Socket To Me

To support servers with multiple clients, the socket interface has a function `select`, that manages a set of sockets.

Invoking this function returns information about what sockets have a packet waiting to be received and which are available for sending a packet.

Proper use of `select` eliminates polling and busy-waiting in a situation where delays are unpredictable (which is always the case with the network).

Spooling & Reservations

A spool is a buffer for a device, like a printer, that can serve only 1 job at a time.

Unlike disk, where it might read for P_1 now and then write for P_2 immediately afterwards, a printer needs to finish a whole print job before it starts the next.

Printing is the most obvious example, but it is by no means the only one.

The operating system centralizes all communication to the printer and runs it through the spooling system.

I/O Protection

Recall from much earlier the idea of kernel mode and user mode instructions.

We want all user accesses to I/O to be mediated through the operating system, so the OS can check to see if the request is valid.

This helps minimize errors and problems where people do bad things like cancel another process's request so theirs can go first.

Our typical tradeoff: increased safety in exchange for reduced performance.