POSIX Device Files

Adrian Kosmaczewski

2007-07-27

Modern operating systems provide a clear separation of the kernel processes from those running in user space, which prompts the question of how to access I/O devices from user processes, without breaking the above mentioned architectural separation, which guarantees stability, security and performance.

Several approaches are available, depending on the level of abstraction used and the context of the problem. One of the solution is using POSIX Device Files, which indeed provide the same system-call interface for both files and devices. This article will describe the POSIX standard, the POSIX device files, and give a short enumeration of advantages and disadvantages of them.

The POSIX Standard

POSIX stands for "Portable Operating System Interface for Unix", and it is the "collective name of a family of related standards specified by the IEEE to define the application programming interface (API) for software compatible with variants of the Unix operating system." (Wikipedia). Nearly all of today's most important operating systems, in either closed or open source form, are POSIX-compliant to a certain degree: all versions of Microsoft Windows, Apple Mac OS X, OpenVMS and Solaris, for example, are fully POSIX-compliant, while Linux, FreeBSD and Nucleos RTOS are partially compliant (Wikipedia). POSIX Conformance Test Suites (for example the one provided by NIST) are used to determine the level of POSIX-compliance of an operating system.

The importance of the POSIX standard is hard to assess; however, it is widely recognized that it has had an enormous impact in the computing world as we know it today:

The overall Unix market (Figure 1) boasts billions of dollars in revenues, and is projected to continue growing into the future. Major areas for this growth feature high-end systems (data warehousing, servers, and supercomputing) rather than desktop applications. Brian Unter evaluated the impact of Posix standards in this area as responsible for up to 30% of Hewlett-Packard's Unix business.

(Isaak & Johnson, 1998)

The POSIX standard provides the whole industry with a single, enormous and strong foundation, allowing interoperability, benchmarking and standardization, which in turn helps reducing costs, making the whole industry more efficient.

POSIX Device Files

One of the key aspects of the POSIX standard is the definition of a common gateway to access I/O devices from the user space, avoiding a direct and probably dangerous communication with the kernel, and known as POSIX Device Files: "These interfaces enable communication with serial, storage, and network devices through device files. In any UNIX-based system such as BSD, a device file is a special file located in /dev that represents a block or character device such as a terminal, disk drive, or printer. If you know the name of a device file (for example, disk0s2 or mt0) your application can use POSIX functions such as open, read, write, and close to access and control the associated device." (Apple)

What does all of this mean? In short, this means that

Under UNIX, every piece of hardware is a file. To demonstrate this, try view the file /dev/hda

```
less -f /dev/hda
```

/dev/hda is not really a file at all. When you read from it, you are actually reading directly from the first physical hard disk of your machine. /dev/hda is known as a device file, and all of them are stored under the /dev directory.

(Sheer)

Advantages

The main advantage is homogeneity. Using POSIX device files, developers can manipulate, read and write data from external I/O devices without having to directly access the operating system kernel, using a unified programming model, with a standardized set of semantics, which leads to systems designed with elegance and correctness.

Another advantage is tightly related to the concept of "streams", as understood by many higher-level programming languages. Streams allow developers to access devices in an uniform way; C++, Java, the .NET languages, and even dynamic languages such as Ruby or Lisp allow to treat sequences of bytes using this uniform concept:

A stream is an object that can be used with an input or output function to identify an appropriate source or sink of characters or bytes for that operation

(Lisp.org)

Java and .NET have big inheritance trees below the abstract Stream class, handling memory-based, file, network or I/O device-based streams, all of them sharing a similar structure and behavior, this means mainly the capability of being read and / or written in a sequential fashion.

This paradigm, coupled with the use of POSIX device files, allow software architects to create flexible designs, where the streams are treated polymorphically, can be exchanged at runtime as needed, and have lesser dependencies among them.

Disadvantages

The main disadvantage might be the maintainability of the code that uses these abstractions, since that these notation and semantics might not be immediately obvious to a newly graduated programmer.

The most common programming experience, in higher-level abstractions, usually depends on higher abstractions to access devices, for example using the System.IO.SerialPort classes in the .NET programming model:

To reduce the programming effort that is required when working with serial ports, the .NET Compact Framework 2.0 includes the SerialPort class. The SerialPort class provides a simplified abstraction over serial communications ports that provides a number of features that simplify monitoring and configuring serial ports. The serial port also simplifies sending and receiving data with serial portsâ€”including the automatic encoding and decoding of data sent to and received from the port

(Microsoft)

Mac OS X also provides higher-level abstractions for managing devices:

A device interface is a plug-in interface between the kernel and a process in user space. The interface conforms to the plug-in architecture defined by Core Foundation Plug-in Services (CFPlugIn), which, in turn, is compatible with the basics of Microsoft's Component Object Model (COM). In the CFPlugIn model, the kernel acts as the plug-in host with its own set of well-defined I/O Kit interfaces, and the I/O Kit framework provides a set of plug-ins (device interfaces) for applications to use

(Apple)

The abstraction-level problem also arises when porting code from an operating system to another, since a common abstraction to access a CD-ROM drive in Windows (usually "D:") is different from the way that Linux does (usually"/cdrom") or the way that Mac OS X uses (which is using the friendly name of the CD-ROM under the "/Volumes" device). The use of POSIX device files might help avoiding these problems.

Conclusion

Both the advantages and disadvantages of using the same system-call interface are relative to the problem being solved, the available budget and the skills of the engineering team. There are huge advantages to stick to the POSIX standard, but there is a cost in maintainability and readability as well.

References

Apple Developer Connection, “I/O Kit Fundamentals, Architectural Overview: Controlling Devices From Outside the Kernel”, [Internet] http://developer.apple.com/documentation/DeviceDrivers/Conceptual/IOKitFundamentals/ArchitectOvervie (Accessed February 11th, 2007)

Isaak, J.; Johnson, L.; “Posix/Unix standards: foundation for 21st century growth”, Micro, IEEE, Volume 18, Issue 4, July-Aug. 1998 Page(s):88, 87; Digital Object Identifier 10.1109/40.710874

Lisp.org, “System Class STREAM”, [Internet] http://www.lisp.org/HyperSpec/Body/syscla_stream.html (Accessed February 11th, 2007)

Microsoft, “What’s New in the .NET Compact Framework 2.0”, [Internet] http://msdn2.microsoft.com/en-us/library/aa446574.aspx (Accessed February 11th, 2007)

NIST, “PCTS:151-2, POSIX Test Suite”, [Internet] http://www.itl.nist.gov/div897/ctg/posix_form.htm (Accessed February 11th, 2007)

Sheer, P.; “UNIX devices”, [Internet] http://www.cacs.louisiana.edu/~mgr/404/burks/linux/rute/node18.htm (Accessed February 11th, 2007)

Wikipedia, “POSIX”, [Internet] http://en.wikipedia.org/wiki/POSIX (Accessed February 11th, 2007)