Matching Device Drivers with Embedded Hardware

There are different sources, and levels of difficulty, for obtaining device drivers for embedded application. Often these vary with the popularity of the OS or the target peripheral. Finding the best source can result in saving time and money.

by Glenn S. Kubota
Micro/sys

In today's fast-paced world, the length of a design cycle for a product is increasingly critical to the projects' success, whether that project is deployed in-house or is a marketable product. In both cases, the adage, time is money holds true. In an embedded system, the operating system and the hardware interface with each other through device drivers. The selection of any piece—operating system, hardware or device driver—impacts the other pieces.

In embedded systems consisting of both hardware and software components, the division of time has shifted from years past. Much of the complexity of todays embedded systems lies in the software. Some systems have multiple programmer-years invested in development. Thus, it is wise to consider ways of reducing some of the time spent on software development. In embedded systems, much of the highly detailed, difficult development is spent on interfacing the software with the hardware. This is where a device driver comes in.

Proper selection of device drivers takes into account the operating system and the hardware, while also balancing the overall performance needs and the availability of each of these pieces. Of course, there are always tradeoffs. A general-purpose operating system may provide quick development time but may require more expensive hardware than a smaller, more optimized operating system. Choosing hardware without any performance cushion may not allow for future enhancements. Choosing unique hardware may mean that device drivers are not available for the selected operating system.

A device driver is code that enables an operating system to interface with a hardware device. In essence, the driver acts as a translator between the operating system and the hardware. This way, the operating system is spared from having to know the details of every piece of hardware to which it may connect.

Device driver development may be quite complex since it requires a register-level understanding of the hardware as well as a low-level understanding of the operating system. For example, a data acquisition board that acquires data through an analog-to-digital converter and then

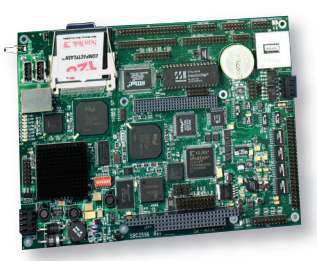

Figure 1 A single board, such as this Micro/sys SBC2596, has three kinds of Linux drivers available: OS-supplied drivers for the serial and parallel ports, compact flash, USB and Ethernet. There are manufacturer-supplied drivers for A/D and D/A I/O and digital I/O. Finally, there is direct hardware access to the global positioning system.

	Advantages	Disadvantages
OS-Supplied Drivers	No need to search for compatible drivers	Only the most popular hardware is supported
Manufacturer-Supplied Drivers	Availability reduces development time	May not be optimal for a specific application
Direct Hardware Access	Very fast access. Simple to use.	May not take advantage of operating system features
Third-Party Proprietary Driver	May add enhanced functionality	Not available for all operating systems or hardware
Roll Your Own Driver	May be highly optimized for the application	May require considerable development and testing time

Table 1 Summary of sources for embedded device drivers.

transfers this data using direct memory access (DMA) cycles, will require a device driver that is capable of initializing the registers of the data acquisition system. It will also need to be capable of setting up the DMA controller through means that are permitted by the operating system. A summary of the possible sources for device drivers is given in Table 1.

OS-Supplied Drivers

Some hardware is so common that many operating systems already come with device drivers as part of the standard distribution. This is especially true of the "embedded PC" architecture based on the venerable IBM PC/XT/AT. Because of its long history in the desktop world, the inclusion of operating system device drivers for many common hardware devices has become de rigueur. Serial ports, printer ports, disk drive controllers, mice, keyboards and video adapters are among the commonly seen devices with native operating system drivers. This greatly benefits the developer because these drivers are probably going to end up in many different projects, and having them ready to go means that they will save a great deal of time each time they are used.

The use of OS-supplied drivers will depend on the hardware selected. The use of unique hardware, such as serial ports that support many different protocols, may increase the performance, but it would most likely eliminate the possibility of using OS-supplied drivers. The advantages and disadvantages need to be examined carefully to determine the best course of action.

The selection of the operating system will also determine the availability of drivers for any particular devices. There are many different devices, so every operating system has a different list of supported hardware.

Manufacturer-Supplied Hardware Drivers

Although many modern operating systems have a wealth of device drivers built in, there are many more hardware devices that do not have built-in support. For example, in a control system, the hardware required may span analog data acquisition boards, motor controllers, digital I/O interfaces and many other unique devices. It is unlikely that an operating system would have support for any of these devices. In this case, the manufacturer of the hardware is probably the most common source for these device drivers.

The reason for getting these device drivers from the hardware manufacturer is the manufacturer is probably the one most familiar with the inner workings of a particular device. It is possible that other software developers could create a more optimized driver for that hardware, but in most cases, this does not make sense from the standpoint of completing the project in a timely fashion. The hardware manufacturer has taken on the tasks of creating a software API (application programming interface), developing code to interface with the hardware, developing code to interface with the operating system, developing sample programs for the device driver, and testing the resultant device driver.

Another criteri on that must be checked when selecting a device driver is whether it is compatible with the specific operating system. There are many different operating systems and it is impossible to support all of them. Generally speaking, the more popular the operating system, the more likely it is that the manufacturer can supply drivers. Note, however, operating systems popular in embedded systems are often quite different from those popular in desktop computer systems.

Direct Hardware Access Drivers

Manufacturer-supplied device drivers are typically separate from the example programs. However, if the operating system allows direct access to the hardware, some devices may be simple enough to access by integrating direct calls to the hardware into the program. For example, under DOS (MS-DOS and other embedded DOS variants), the application program can access the hardware without any special permissions. Generally speaking, many digital I/O boards can be accessed with reads and writes to a few ports. These accesses may be part of the program, or a few primitive routines may be used to add one level of abstraction.

One of the advantages of directly reading and writing to the hardware is that it allows the fastest possible accesses. In some time-critical applications this may be important. Another advantage is that source code availability makes for a greater understanding of the internals of the device and also may allow porting to other operating systems.

There are many different operating systems and pieces of hardware where this type of driver makes sense. It may be a question of whether this allows access to the needed parts of the device. Also, in some cases, this may be the only software support for a particular device.

Open-Source Drivers

With the increasing popularity of the open-source movement, particularly in the area of Linux, open-source device drivers developed by non-vendors are becoming more common. In some cases these device drivers were developed from published hardware specifications, and in other cases the hardware interface was reverse-engineered. Open-source device drivers are often of high quality because many people have access to the source code to check for bugs. Also, the availability of the source code allows customization for a particular application.

The main problem with non-vendor drivers for embedded systems is that they tend to be more available for more popular hardware. For example, an industrial control and monitoring panel would probably be able to use non-vendor touch screen drivers, but non-vendor drivers for a particular data acquisition board would probably not be available.

Third-Party Proprietary Drivers

Occasionally there are device drivers available that are not from the manufacturer and are not open-source. Often these drivers are available to provide some enhanced functionality. For example, a CANbus driver from a manufacturer may offer low-level access to the CANbus, but there could be a third-party device driver that can communicate with some higher-level protocol, such as DeviceNet or CANopen.

The availability of third-party device drivers will depend on both the operating system and the hardware selected. Again, planning ahead is important for determining whether all of these pieces will work well together.

Roll Your Own Driver

Although not the preferred method due to time constraints, there are a few instances where creating a custom driver may still be necessary. Unique operating systems or custom hardware are such cases. For example, there are many real-time operating systems available. Only a few of the larger ones are likely to have driver support for any particular hardware. And of course, in the case of a full custom designed I/O board, there will not be drivers available.

When creating a brand-new device driver for an unsupported operating system, it is still advantageous to have driver source code for a supported operating system. Even though the operating system calls are likely to be quite different, the code that communicates with the registers of the hardware is going to be similar. Plenty of time must be available to create and test a custom driver. It can be tricky to debug because of the asynchronous nature of the hardware.

There are different ways of getting device drivers for embedded systems, each with their own advantages and disadvantages. And the same system may have drivers from several different sources (Figure 1). There are many tradeoffs in selecting the hardware, operating system and device drivers. The process of selection is iterative. Selecting any one piece without considering the effect on the other pieces is a recipe for disaster. In general, having the drivers early in the design cycle will make the design go more quickly. However, itís essential that the drivers are actually compatible with the particular operating system and hardware being used. Device drivers are the glue between the operating system and the hardware. A little attention can pay off in a big way. ◢

Micro/sys
Montrose, CA.
(818) 244-4600.
[www.embeddedsys.com].