NAME

Intro, intro – introduction to device driver interfaces

DESCRIPTION

Section 9 provides reference information needed to write device drivers for Solaris 2.x. It describes the interfaces provided by the Device Driver Interface Driver-Kernel Interface (DDI/DKI). Drivers that conform to this specification are more likely to work in future releases and may be portable to other environments.

Porting

Software is usually considered portable if it can be adapted to run in a different environment more cheaply than it can be rewritten. The new environment may include a different processor, operating system, and even the language in which the program is written, if a language translator is available. Likewise the new environment might include multiple processors. More often, however, software is ported between environments that share an operating system, processor, and source language. The source code is modified to accommodate the differences in compilers or processors or releases of the operating system.

In the past, device drivers did not port easily for one or more of the following reasons:

- To enhance functionality, members had been added to kernel data structures accessed by drivers, or the sizes of existing members had been redefined.
- The calling or return syntax of kernel functions had changed.
- Driver developers did not use existing kernel functions where available, or relied on undocumented side effects that were not maintained in the next release.
- Architecture-specific code had been scattered throughout the driver when it could have been isolated.

Operating systems are periodically reissued to customers as a way to improve performance, fix bugs, and add new features. This is probably the most common threat to compatibility encountered by developers responsible for maintaining software. Another common problem is upgrading hardware. As new hardware is developed, customers occasionally decide to upgrade to faster, more capable computers of the same family. Although they may run the same operating system as those being replaced, architecture-specific code may prevent the software from porting.

Scope of Interfaces

Although application programs have all of the porting problems mentioned, developers attempting to port device drivers have special challenges. Before describing the DDI/DKI, it is necessary to understand the position of device drivers in operating systems.

Device drivers are kernel modules that control data transferred to and received from peripheral devices but are developed independently from the rest of the kernel. If the goal of achieving complete freedom in modifying the kernel is to be reconciled with the goal of binary compatibility with existing drivers, the interaction between drivers and the kernel must be rigorously regulated. This driver/kernel service interface is the most important of the three distinguishable interfaces for a driver, summarized as follows:

- Driver–Kernel. I/O System calls result in calls to driver entry point routines. These make up the kernel-to-driver part of the service interface, described in Section 9E. Drivers may call any of the functions described in Section 9F. These are the driver-to-kernel part of the interface.

- Driver–Hardware. All drivers (except software drivers) must include code for interrupt handling, and may also perform direct memory access (DMA). These and other hardware-specific interactions make up the driver/hardware interface.
- Driver–Boot/Configuration Software. The interaction between the driver and the boot and configuration software is the third interface affecting drivers.

Scope of the DDI/DKI

The primary goal of the DDI/DKI is to facilitate both source and binary portability across successive releases of the operating systems on a particular machine. In addition, it promotes source portability across implementations of UNIX on different machines, and applies only to implementations based on System V Release 4. The DDI/DKI consists of several sections:

- DDI/DKI Architecture Independent - These interfaces are supported on all implementations of System V Release 4, and will be supported in future releases of System V.
- DKI-only - These interfaces are part of System V Release 4, and may not be supported in future releases of System V. There are only two interfaces in this class, **segmap**(9E) and **hat_getkpfnum**(9F)**.**
- Solaris DDI - These interfaces specific to Solaris, and will be supported in future releases of Solaris 2.x.
- Solaris SPARC specific DDI - These interfaces are specific to the SPARC processor, and may not be available on other processors supported by Solaris.
- Solaris x86 specific DDI - These interfaces are specific to the x86 processor, and may not be available on other processors supported by Solaris.

To achieve the goal of source and binary compatibility, the functions, routines, and structures specified in the DDI/DKI must be used according to these rules.

- Drivers cannot access system state structures (for example, **u** and **sysinfo**) directly.
- For structures external to the driver that may be accessed directly, only the utility functions provided in Section 9F should be used. More generally, these functions should be used wherever possible.
- The headers **<sys/ddi.h>** and **<sys/sunddi.h>** must be the last header files included by the driver.

Audience

Section 9 is for software engineers responsible for creating, modifying, or maintaining drivers that run on this operating system and beyond. It assumes that the reader is familiar with system internals and the C Programming Language.

How to Use Section 9

Section 9 is divided into three subsections:

9E Driver Entry Points – contains reference pages for all driver entry point routines.

9F Kernel Functions – contains reference pages for all driver support routines.

9S Data Structures – contains reference pages for driver-related structures.

SEE ALSO

intro(9E), **intro**(9F), **intro**(9S)

NOTES

SunSoft's implementation of the DDI/DKI was designed to provide binary compatibility for third-party device drivers across currently supported hardware platforms across minor releases of the operating system.

However, unforeseen technical issues may force changes to the binary interface of the DDI/DKI. We cannot therefore promise or in any way assure that DDI/DKI-compliant device drivers will continue to operate correctly on future releases.

Furthermore, future releases may contain additions to the DDI/DKI to support future platforms. At that time device drivers wishing to operate across the new set of supported platforms may require these additions.