Accesos a Dispositivos Independientes del Bus

Matthew Wilcox

matthew@wil.cx

Alan Cox

alan@redhat.com

Accesos a Dispositivos Independientes del Bus

por Matthew Wilcox

por Alan Cox

Tabla de contenidos

Capítulo 1. Introducción

Linux suministra una API que abstrae la realización de E/S a través de todos los buses y dispositivos, permitiendo a los controladores de dispositivos ser escritos independientemente del tipo de bus.

Capítulo 2. Bugs Conocidos Y Suposiciones

Ninguno.

Capítulo 3. E/S Mapeada en Memoria

3.1. Obteniendo Acceso al Dispositivo

La forma más ampliamente soportada de E/S es la E/S por mapeo de memoria. Esto es, una parte del espacio de direcciones de la CPU es interpretada no como un acceso a la memoria, sino como un acceso a un dispositivo. Algunas arquitecturas definen dispositivos para estar en una dirección fija, pero la mayoría tiene algún método para descubrir los dispositivos. El bus PCI es un buen ejemplo de este esquema. Este documento no cubre cómo recibir una dirección, sino que asume que ya estás empezando en una. Las direcciones físicas son del tipo unsigned long.

Estas direcciones no deberían de ser usadas directamente. En vez de esto, para obtener una dirección utilizable para pasar a las funciones de acceso descritas posteriormente, debería de llamar a `ioremap`. Te será devuelta una dirección utilizable para el acceso del dispositivo.

Después de que hayas finalizado de usar el dispositivo (esto es, en la rutina de salida en tu módulo), llama a `iounmap` para retornar el espacio de direcciones al núcleo. La mayoría de las arquitecturas asignan un nuevo espacio de direcciones cada vez que llamas y haces `ioremap`, y pueden acabarse a menos que llames a `iounmap`.

3.2. Accediendo al dispositivo

La parte de la interface más usada por los controladores es la lectura y escritura de los registros mapeados en memoria del dispositivo. Linux suministra interfaces para leer y escribir cantidades de 8-bits, 16-bits, 32-bits y 64-bits. Debido a un accidente histórico estos son llamados accesos byte, word, long y quad. Ambos accesos de lectura y escritura son soportados; no hay soporte pre-producido en este momento.

Las funciones tienen los nombres `readb`, `readw`, `readl`, `readq`, `writeb`, `writew`, `writel` y `writeq`.

A algunos dispositivos (como los framebuffers) les gustaría usar transferencias más grandes de los 8 bytes cada vez. Para estos dispositivos, son suministradas las funciones `memcpy_toio`, `memcpy_fromio` y `memset_io`. No utilices memset o memcpy en direcciones de E/S; no está garantizado que copien los datos en orden.

Las funciones de lectura y escritura están definidas para ser ordenadas. Esto es, al compilador no le está permitido reordenar la secuencia de E/S. Cuando el órden puede ser compilado de forma optimizada, puedes usar `__readb` y amigos para indicar un órden relajado. Usa esto con cuidado. La `rmb` suministra una barrera de lectura de memoria. La `wmb` suministra una barrera de escritura de memoria.

Mientras las funciones básicas son definidas para ser síncronas y ordenadas con respecto a las otras, los

dispositivos que están en los buses quizás sean asíncronos. En particular, algunos autores han sido quemados por el hecho de que las escrituras en el bus PCI son realizadas de forma asíncrona. El autor de un controlador debe de emitir una lectura para el mismo dispositivo para asegurarse de que la escritura ha tenido lugar en los casos específicos que quiere el autor. Este tipo de propiedad no puede ser escondida por los escritores de los controladores en la API.

3.3. Funciones de Herencia ISA

En los núcleos viejos (2.2 y anteriores) el bus ISA podía leer o escribir con estas funciones y sin usar ioremap. Esto ya no es verdad en Linux 2.4. Un conjunto de funciones equivalentes existen para el cambio de controladores con herencia fácil. Las funciones disponibles son prefijadas con 'isa_' y son `isa_readb`, `isa_writeb`, `isa_readw`, `isa_writew`, `isa_readl`, `isa_writel`, `isa_memcpy_fromio` y `isa_memcpy_toio`

Estas funciones no deberían de ser usadas en los nuevos controladores, y serán quitadas eventualmente.

Capítulo 4. Accesos al Espacio de Puerto

4.1. Espacio de Puerto Explicado

Otra forma de E/S comúnmente usada en el Espacio de Puerto. Esta es un rango de direcciones separados del espacio normal de direcciones de memoria. El acceso a estas direcciones no es generalmente tan rápido como los accesos a las direcciones mapeadas en memoria, y por lo tanto tienen potencialmente un espacio de direcciones más pequeño.

A diferencia de la E/S mapeada en memoria, no se requiere preparación para acceder al espacio de puerto.

4.2. Accediendo al Espacio de Puerto

Los accesos a este espacio son suministrados a través de un conjunto de funciones que permiten accesos de 8-bits, 16-bits y 32-bits; también conocidos como byte, word y long. Estas funciones son `inb`, `inw`, `inl`, `outb`, `outw` y `outl`.

Son suministradas algunas variantes para estas funciones. Algunos dispositivos requieren que los accesos a sus puertos se realicen más despacio. Esta funcionalidad es suministrada añadiendo una `_p` al final de la función. También hay equivalentes a memcpy. Las funciones `ins` y `outs` copian bytes, words o longs al puerto dado.

Capítulo 5. Funciones Públicas Suministradas

Capítulo 6. Sobre la traducción

Este documento es la traducción de "Bus-Independent Device Accesses", documento que acompaña al código del núcleo de Linux, versión 2.4.18.

Este documento ha sido traducido por Rubén Melcón <melkon@terra.es>; y es publicado por el Proyecto Lucas (http://lucas.hispalinux.es)

Versión de la tradución 0.04 (Julio de 2002).

Si tienes comentarios sobre la traducción, ponte en contacto con Rubén Melcón <melkon@terra.es>