EBook Gratis

APRENDIZAJE linux-kernel

Free unaffiliated eBook created from
Stack Overflow contributors.

#linux-kernel

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: linux-kernel

It is an unofficial and free linux-kernel ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official linux-kernel.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con linux-kernel

Observaciones

Esta sección proporciona una descripción general de qué es linux-kernel y por qué un desarrollador puede querer usarlo.

También debe mencionar cualquier tema grande dentro de linux-kernel, y vincular a los temas relacionados. Dado que la Documentación para linux-kernel es nueva, es posible que deba crear versiones iniciales de esos temas relacionados.

Versiones

Versión	Fecha de lanzamiento
4.4	2016-01-10
4.1	2015-06-21
3.18	2014-12-07
3.16	2014-08-03
3.12	2013-11-03
3.10	2013-06-30
3.4	2012-05-20
3.2	2012-01-04

Examples

Instalación o configuración

El código fuente del kernel de Linux se puede encontrar en https://www.kernel.org/

Descargar extraer e ingresar al directorio del kernel

Escriba estos comandos paso a paso en su terminal. (Elija la versión adecuada que necesita en lugar de linux-4.7.tar.gz)

```
wget http://www.kernel.org/pub/linux/kernel/v4.7/linux-4.7.tar.gz
tar zxvf linux-4.7.tar.gz
cd linux-4.7
```

`make menuconfig` seleccionará las características requeridas para el kernel. Las configuraciones antiguas del kernel se pueden copiar utilizando el archivo `.config` antiguo y ejecutando `make oldconfig` . También podemos usar `make xconfig` como una versión gráfica de la herramienta de configuración.

Construye las dependencias, compila el kernel y los módulos.

```
make dep
make bzImage
make modules
make modules_install
```

Alternativamente, si desea reconfigurar el núcleo antiguo y volver a compilarlo, ejecute los siguientes comandos:

```
make mrproper
make menuconfig
make dep
make clean
make bzImage
make modules
make modules_install
```

A continuación, copie el núcleo, `system.map` archivo a `/boot/vmlinuz-4.7`

crear un archivo `.conf` con el contenido a continuación

```
image = /boot/vmlinuz-4.7
label = "Linux 4.7"
```

Luego ejecute `lilo -v` para modificar el sector de arranque y reiniciar.

Lea Empezando con linux-kernel en línea: https://riptutorial.com/es/linux-kernel/topic/2385/empezando-con-linux-kernel

Capítulo 2: Cómo encontrar a la persona adecuada para la ayuda.

Introducción

Esto debería reflejar algunos de los documentos oficiales del kernel de Linux y publicar enlaces a las últimas versiones de dichos documentos en `tovalds/linux` en GitHub.com. La idea es alentar a las personas a utilizar los archivos de `MAINTAINERS` , la lista de correo de `linux-kernel` , el `git log` y los `scripts/get-maintainer` , para que estén familiarizados con las formas más comunes de identificar un punto de contacto clave.

Examples

Encuentre los mantenedores "probables" para el convertidor serial USB FTDI

Primero, determine el archivo fuente para este controlador en particular. Lo encontré en `drivers/usb/serial/ftdi_sio.c` .

```
./scripts/get_maintainer.pl drivers/usb/serial/ftdi_sio.c
```

Y los resultados:

```
Johan Hovold <johan@kernel.org> (maintainer:USB SERIAL SUBSYSTEM)
Greg Kroah-Hartman <gregkh@linuxfoundation.org> (supporter:USB SUBSYSTEM)
linux-usb@vger.kernel.org (open list:USB SERIAL SUBSYSTEM)
linux-kernel@vger.kernel.org (open list)
```

Ahora sabemos a quién pedir ayuda con este controlador en particular, y qué direcciones de correo electrónico deben enviarse por CC al enviar un parche contra este controlador.

Lea Cómo encontrar a la persona adecuada para la ayuda. en línea: https://riptutorial.com/es/linux-kernel/topic/10056/como-encontrar-a-la-persona-adecuada-para-la-ayuda-

Capítulo 3: Controlador del dispositivo Linux Hello World

Examples

Un módulo vacío del núcleo

```
#include <linux/init.h>
#include <linux/module.h>

/**
 * This function is called when the module is first loaded.
 */
static int __init hello_kernel_init(void)
{
    printk("Hello, World!\n");
    return 0;
}

/**
 * This function is called when is called if and when the module is unloaded.
 */
static void __exit hello_kernel_exit(void)
{
    printk("Goodbye, cruel world...\n");
}

/* The names of the init/exit functions are arbitrary, and they are bound using the following
macro definitions */
module_init(hello_kernel_init);
module_exit(hello_kernel_exit);
```

Para escribir un controlador de dispositivo Linux (dispositivo de caracteres, dispositivo de bloque, etc.), es necesario crear un módulo de kernel que tenga puntos de entrada y salida.

Por sí mismo, el módulo kernel no hace nada; no tiene una manera significativa de comunicarse con el espacio de usuario. Usando el punto de entrada, es posible crear un nuevo dispositivo de carácter, por ejemplo, que luego se utiliza para comunicarse con el espacio de usuario.

Construyendo y ejecutando el módulo.

Para compilar el controlador, es necesario tener el árbol de origen del kernel de Linux.

Suponiendo que las fuentes están en `/lib/modules/<kernel-version>` , el siguiente Makefile compilará el archivo `driver.c` en el `driver.ko` kernel `driver.ko`

```
obj-m := driver.o
KDIR := /lib/modules/$(shell uname -r)/build/
PWD := $(shell pwd)
```

```
all:
    $(MAKE) -C $(KDIR) M=$(PWD) modules
```

Observe cómo se `make` estas llamadas de Makefile en el directorio de compilación del Kernel.

Cuando el paso de compilación finalice satisfactoriamente, el directorio src del controlador tendrá un aspecto similar al siguiente:

```
driver.c  driver.ko  driver.mod.c  driver.mod.o  driver.o  Makefile  modules.order
Module.symvers
```

Para "ejecutar" el módulo, es necesario insertar en el kernel en ejecución:

```
$ insmod driver.ko
$ dmesg | tail -n 1
[133790.762185] Hello, World!

$ rmmod driver.ko
$ dmesg | tail -n 1
[133790.762185] Goodbye, cruel world...
```

Lea Controlador del dispositivo Linux Hello World en línea: https://riptutorial.com/es/linux-kernel/topic/7056/controlador-del-dispositivo-linux-hello-world

Capítulo 4: Creación y uso de hilos de kernel.

Introducción

Este tema trata sobre cómo crear y usar hilos del kernel.

Examples

Creación de hilos del núcleo.

kern_thread.c

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/kthread.h>
#include <linux/sched.h>

#define AUTHOR        "Nachiket Kulkarni"
#define DESCRIPTION    "Simple module that demonstrates creation of 2 kernel threads"

static int kthread_func(void *arg)
{
/* Every kthread has a struct task_struct associated with it which is it's identifier.
* Whenever a thread is schedule for execution, the kernel sets "current" pointer to
* it's struct task_struct.
* current->comm is the name of the command that caused creation of this thread
* current->pid is the process of currently executing thread
*/
    printk(KERN_INFO "I am thread: %s[PID = %d]\n", current->comm, current->pid);
    return 0;
}

static int __init init_func(void)
{
    struct task_struct *ts1;
    struct task_struct *ts2;
    int err;

    printk(KERN_INFO "Starting 2 threads\n");

/*struct task_struct *kthread_create(int (*threadfn)(void *data), void *data, \
 *                          const char *namefmt, ...);
 * This function creates a kernel thread and starts the thread.
 */
    ts1 = kthread_run(kthread_func, NULL, "thread-1");
    if (IS_ERR(ts1)) {
        printk(KERN_INFO "ERROR: Cannot create thread ts1\n");
        err = PTR_ERR(ts1);
        ts1 = NULL;
        return err;
    }

    ts1 = kthread_run(kthread_func, NULL, "thread-1");
```

```
    if (IS_ERR(ts1)) {
        printk(KERN_INFO "ERROR: Cannot create thread ts1\n");
        err = PTR_ERR(ts1);
        ts1 = NULL;
        return err;
    }

    printk(KERN_INFO "I am thread: %s[PID = %d]\n", current->comm, current->pid);
    return 0;
}

static void __exit exit_func(void)
{
    printk(KERN_INFO "Exiting the module\n");
}

module_init(init_func);
module_exit(exit_func);

MODULE_AUTHOR(AUTHOR);
MODULE_DESCRIPTION(MODULE_AUTHOR);
MODULE_LICENSE("GPL");
```

Makefile

```
obj-m += kern_thread.o

all:
    make -C /lib/module/$(shell uname -r)/build M=$(PWD) modules
clean:
    make -C /lib/module/$(shell uname -r)/build M=$(PWD) clean
```

Al insertar el .ko, se imprimió:

```
Starting 2 threads
I am thread: thread-1[PID = 6786]
I am thread: insmod[PID = 6785]
I am thread: thread-2[PID = 6788]
```

Lea Creación y uso de hilos de kernel. en línea: https://riptutorial.com/es/linux-kernel/topic/10619/creacion-y-uso-de-hilos-de-kernel-

Capítulo 5: Linux: Named Pipes (FIFO)

Examples

Lo que se llama Pipe (FIFO)

```
A named pipe is really just a special kind of file (a FIFO file) on the local hard drive.
Unlike a regular file, a FIFO file does not contain any user information. Instead, it allows
two or more processes to communicate with each other by reading/writing to/from this file.

A named pipe works much like a regular pipe, but does have some noticeable differences.

Named pipes exist as a device special file in the file system.
Processes of different ancestry can share data through a named pipe.
When all I/O is done by sharing processes, the named pipe remains in the file system for later
use.

The easiest way to create a FIFO file is to use the mkfifo command. This command is part of
the standard Linux utilities and can simply be typed at the command prompt of your shell. You
may also use the mknod command to accomplish the same thing.
```

Lea Linux: Named Pipes (FIFO) en línea: https://riptutorial.com/es/linux-kernel/topic/6144/linux--named-pipes--fifo-

Capítulo 6: Llamada al sistema de horquilla

Examples

tenedor () llamada al sistema

`fork()` es una llamada al sistema. fork se utiliza para crear un proceso hijo a partir del proceso en ejecución, que es una réplica del proceso padre (el proceso que ejecutó `fork()`). El proceso hijo se deriva del proceso padre. Tanto el padre como el hijo tienen un espacio de dirección diferente, cada uno es independiente de los cambios realizados en las variables.

El proceso hijo tiene su propio PID (identificación de proceso). El PPID (ID del proceso principal) del proceso secundario es el mismo que el PID del proceso principal.

> Formato:
>
> Archivo de encabezado: `#include <unistd.h>`
> Declaración de la función: `pid_t fork(void);`

fork () no necesita ningún argumento de entrada.

Al crear con éxito el proceso hijo, el pid del proceso hijo se devuelve al proceso padre y se devuelve 0 al proceso hijo. En caso de error devuelve `-1` sin proceso creado.

Ejemplo de uso:

```
#include <stdio.h>
#include <unistd.h>

void child_process();
void parent_process();

int main()
{
    pid_t pid;
    pid=fork();
    if(pid==0)
        child_process();
    else
        parent_process();
    return 0;
}

/*getpid() will return the Pid of the
  current process executing the function */

void child_process()
{
    printf("Child process with PID : %d  and PPID : %d ", getpid(),getppid());
}

void parent_process()
```

```
{
    printf("Parent process with PID : %d", getpid());
}
```

La secuencia de las declaraciones de `printf` del hijo y del padre depende del mecanismo de programación que depende únicamente del sistema.

Lea Llamada al sistema de horquilla en línea: https://riptutorial.com/es/linux-kernel/topic/5199/llamada-al-sistema-de-horquilla

Capítulo 7: Seguimiento de eventos

Examples

Rastreo de eventos I2C

Nota: Supongo que `debugfs` se monta en `/sys/kernel/debug`

Si no, prueba:

```
mount -t debugfs none /sys/kernel/debug
```

Cambie al directorio de seguimiento:

```
cd /sys/kernel/debug/tracing/
```

Asegúrese de que la función de trazador está deshabilitada:

```
echo nop > current_tracer
```

Habilitar todos los eventos I2C:

```
echo 1 > events/i2c/enable
```

Asegúrese de que el rastreo esté habilitado:

```
echo 1 > tracing_on
```

Los mensajes de seguimiento se pueden ver en `/sys/kernel/debug/tracing/trace` , ejemplo:

```
... i2c_write: i2c-5 #0 a=044 f=0000 l=2 [02-14]
... i2c_read: i2c-5 #1 a=044 f=0001 l=4
... i2c_reply: i2c-5 #1 a=044 f=0001 l=4 [33-00-00-00]
... i2c_result: i2c-5 n=2 ret=2
```

La documentación de la API del espacio de usuario de trace events se puede encontrar en el archivo `Documentation/trace/events.txt` de la fuente del kernel.

Lea Seguimiento de eventos en línea: https://riptutorial.com/es/linux-kernel/topic/3466/seguimiento-de-eventos

Creditos

S. No	Capítulos	Contributors
1	Empezando con linux-kernel	Community, EsmaeelE, Marek Skiba, Matt, Tejus Prasad, vinay hunachyal
2	Cómo encontrar a la persona adecuada para la ayuda.	DevNull, EsmaeelE
3	Controlador del dispositivo Linux Hello World	Gilad Naaman
4	Creación y uso de hilos de kernel.	nachiketkulk
5	Linux: Named Pipes (FIFO)	chait
6	Llamada al sistema de horquilla	chait, Dilip Kumar
7	Seguimiento de eventos	sergej