Kostenloses eBook

LERNEN linux-kernel

Free unaffiliated eBook created from
Stack Overflow contributors.

#linux-kernel

Inhaltsverzeichnis

Über ... 1

Kapitel 1: Erste Schritte mit dem Linux-Kernel ... 2

Bemerkungen ... 2

Versionen ... 2

Examples ... 2

Installation oder Setup ... 2

Laden Sie das Extrakt herunter und geben Sie das Kernel-Verzeichnis ein ... 2

Erstellen Sie die Abhängigkeiten, kompilieren Sie den Kernel und die Module. ... 3

Kapitel 2: Ereignisverfolgung ... 4

Examples ... 4

Verfolgung von I2C-Ereignissen ... 4

Kapitel 3: Erstellung und Verwendung von Kernel-Threads ... 5

Einführung ... 5

Examples ... 5

Erstellung von Kernel-Threads ... 5

Kapitel 4: Gabel Systemaufruf ... 7

Examples ... 7

Fork () Systemaufruf ... 7

Kapitel 5: Linux: Named Pipes (FIFO) ... 9

Examples ... 9

Was ist Named Pipe (FIFO) ... 9

Kapitel 6: So finden Sie die richtige Person für Hilfe. ... 10

Einführung ... 10

Examples ... 10

Finden Sie die "wahrscheinlichen" Betreuer für den seriellen FTDI USB-Konverter ... 10

Kapitel 7: Treiber für Linux Hello World Device ... 11

Examples ... 11

Ein leeres Kernelmodul ... 11

Das Modul erstellen und ausführen ... 11

Credits.....13

Über

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

Kapitel 1: Erste Schritte mit dem Linux-Kernel

Bemerkungen

Dieser Abschnitt bietet einen Überblick über den Linux-Kernel und warum ein Entwickler ihn verwenden möchte.

Es sollte auch alle großen Themen innerhalb des Linux-Kernels erwähnen und auf die verwandten Themen verweisen. Da die Dokumentation für Linux-Kernel neu ist, müssen Sie möglicherweise erste Versionen dieser verwandten Themen erstellen.

Versionen

Ausführung	Veröffentlichungsdatum
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

Installation oder Setup

Linux-Kernel-Quellcode finden Sie unter https://www.kernel.org/

Laden Sie das Extrakt herunter und geben Sie das Kernel-Verzeichnis ein

Geben Sie diese Befehle schrittweise in Ihr Terminal ein (Wählen Sie anstelle von linux-4.7.tar.gz die gewünschte Version aus.)

```
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` wählt die für den Kernel erforderlichen Funktionen aus. Alte `.config` können mithilfe der alten `.config` Datei und der Ausführung von `make oldconfig` . Außerdem können wir `make xconfig` als grafische Version des Konfigurationstools verwenden.

Erstellen Sie die Abhängigkeiten, kompilieren Sie den Kernel und die Module.

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

Wenn Sie den alten Kernel neu konfigurieren und neu kompilieren möchten, führen Sie die folgenden Befehle aus:

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

Kopieren Sie dann den Kernel, `system.map` Datei `/boot/vmlinuz-4.7`

Erstellen Sie eine `.conf` Datei mit dem folgenden Inhalt

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

`lilo -v` dann `lilo -v` , um den Bootsektor zu ändern und einen Neustart durchzuführen.

Erste Schritte mit dem Linux-Kernel online lesen: https://riptutorial.com/de/linux-kernel/topic/2385/erste-schritte-mit-dem-linux-kernel

Kapitel 2: Ereignisverfolgung

Examples

Verfolgung von I2C-Ereignissen

Anmerkung: Ich `debugfs` davon aus, dass `debugfs` unter `/sys/kernel/debug` eingehängt ist

Wenn nicht, probiere es aus:

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

Wechseln Sie in das Tracing-Verzeichnis:

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

Stellen Sie sicher, dass der Funktionstracer deaktiviert ist:

```
echo nop > current_tracer
```

Alle I2C-Ereignisse aktivieren:

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

Stellen Sie sicher, dass die Ablaufverfolgung aktiviert ist:

```
echo 1 > tracing_on
```

Die Trace-Nachrichten können in `/sys/kernel/debug/tracing/trace` angezeigt werden, Beispiel:

```
... 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
```

Die Trace-User-Space-API-Dokumentation befindet sich in der Datei `Documentation/trace/events.txt` der `Documentation/trace/events.txt` .

Ereignisverfolgung online lesen: https://riptutorial.com/de/linux-kernel/topic/3466/ereignisverfolgung

Kapitel 3: Erstellung und Verwendung von Kernel-Threads

Einführung

In diesem Thema wird beschrieben, wie Kernel-Threads erstellt und verwendet werden.

Examples

Erstellung von Kernel-Threads

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
```

Beim Einfügen der .ko wurde Folgendes gedruckt:

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

Erstellung und Verwendung von Kernel-Threads online lesen: https://riptutorial.com/de/linux-kernel/topic/10619/erstellung-und-verwendung-von-kernel-threads

Kapitel 4: Gabel Systemaufruf

Examples

Fork () Systemaufruf

`fork()` ist ein Systemaufruf. Mit fork wird ein untergeordneter Prozess aus dem laufenden Prozess erstellt. Hierbei handelt es sich um eine Replik des übergeordneten Prozesses (Prozess, der `fork()`). Der untergeordnete Prozess wird vom übergeordneten Prozess abgeleitet. Sowohl das übergeordnete Element als auch das untergeordnete Element haben unterschiedliche Adressräume. Jeder ist unabhängig von den an den Variablen vorgenommenen Änderungen.

Der Kindprozess hat eine eigene PID (Prozessidentifikation). PPID (Parent Process ID) des untergeordneten Prozesses entspricht der PID des übergeordneten Prozesses.

> Format:
>
> Header-Datei: `#include <unistd.h>`
> Funktionsdeklaration: `pid_t fork(void);`

fork () benötigt keine Eingabeargumente.

Bei erfolgreicher Erstellung des untergeordneten Prozesses wird die PID des untergeordneten Prozesses an den übergeordneten Prozess und 0 im untergeordneten Prozess zurückgegeben. Bei einem Fehler wird `-1` ohne dass ein Prozess erstellt wurde.

Anwendungsbeispiel:

```
#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());
}
```

Die Reihenfolge der `printf` Anweisungen von Child und Parent hängt von dem Scheduling-Mechanismus ab, der rein systembedingt ist.

Gabel Systemaufruf online lesen: https://riptutorial.com/de/linux-kernel/topic/5199/gabel-systemaufruf

Kapitel 5: Linux: Named Pipes (FIFO)

Examples

Was ist Named 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.
```

Linux: Named Pipes (FIFO) online lesen: https://riptutorial.com/de/linux-kernel/topic/6144/linux--named-pipes--fifo-

Kapitel 6: So finden Sie die richtige Person für Hilfe.

Einführung

Dies sollte einige der offiziellen Linux-Kernel-Dokumente `tovalds/linux` und Links zu den neuesten Versionen der genannten Dokumente in `tovalds/linux` auf GitHub.com bereitstellen. Die Idee besteht darin, Einzelpersonen dazu zu ermutigen, die `MAINTAINERS` Dateien, die Mailing-Liste für `linux-kernel` , das `git log` und die `scripts/get-maintainer` , damit sie mit den üblicherweise verwendeten Methoden zur Identifizierung einer zentralen Kontaktstelle vertraut sind.

Examples

Finden Sie die "wahrscheinlichen" Betreuer für den seriellen FTDI USB-Konverter

Bestimmen Sie zuerst die Quelldatei für diesen bestimmten Treiber. Fand es unter `drivers/usb/serial/ftdi_sio.c` .

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

Und die Ergebnisse:

```
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)
```

Jetzt wissen wir, wer mit diesem Treiber Hilfe anpingen soll und welche E-Mail-Adressen beim Senden eines Patches gegen diesen Treiber mit CC-Adressen versehen werden sollen.

So finden Sie die richtige Person für Hilfe. online lesen: https://riptutorial.com/de/linux-kernel/topic/10056/so-finden-sie-die-richtige-person-fur-hilfe-

Kapitel 7: Treiber für Linux Hello World Device

Examples

Ein leeres Kernelmodul

```
#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);
```

Um einen Linux-Gerätetreiber (Character-Device, Block-Device usw.) schreiben zu können, müssen Sie ein Kernel-Modul erstellen, das über Ein- und Ausfahrtpunkte verfügt.

Das Kernelmodul alleine macht nichts; Es gibt keine sinnvolle Möglichkeit, mit dem Benutzerraum zu kommunizieren. Über den Einstiegspunkt kann zum Beispiel ein neues Zeichengerät erstellt werden, mit dem dann mit dem Benutzerraum kommuniziert wird.

Das Modul erstellen und ausführen

Zum Kompilieren des Treibers ist der Linux-Kernel-Quellbaum erforderlich.

Angenommen, die Quellen sind `/lib/modules/<kernel-version>` . Das folgende Makefile kompiliert die Datei `driver.c` in das Kernel-Objekt `driver.ko`

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

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

Beachten Sie, wie diese Makefile Anrufe `make` im Build - Verzeichnis des Kernel.

Wenn der Kompilierungsschritt erfolgreich abgeschlossen wurde, sieht das src-Verzeichnis des Treibers folgendermaßen aus:

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

Um das Modul "laufen" zu können, müssen Sie Folgendes in den laufenden Kernel einfügen:

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

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

Treiber für Linux Hello World Device online lesen: https://riptutorial.com/de/linux-kernel/topic/7056/treiber-fur-linux-hello-world-device

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit dem Linux-Kernel	Community, EsmaeelE, Marek Skiba, Matt, Tejus Prasad, vinay hunachyal
2	Ereignisverfolgung	sergej
3	Erstellung und Verwendung von Kernel-Threads	nachiketkulk
4	Gabel Systemaufruf	chait, Dilip Kumar
5	Linux: Named Pipes (FIFO)	chait
6	So finden Sie die richtige Person für Hilfe.	DevNull, EsmaeelE
7	Treiber für Linux Hello World Device	Gilad Naaman