FH-OÖ Hagenberg/ESD
Embedded Operating Systems, WS 2014
Rainer Findenig, Florian Eibensteiner, Josef Langer © 2014

2. Übung: Embedded Linux: Remote Controlling

Name(n): Punkte:

1. Aufgabe *Theorie: Analyse der Flash-Partitionen*

In der letzten Übung haben Sie ein auf embedded Linux basierendes System entwickelt, das jedoch ohne den Entwicklungsrechner nicht lauffähig war, da sowohl der Kernel als auch das *Root File System* über das lokale Netzwerk zur Verfügung gestellt wurden. Dies ist zwar während der Entwicklung hilfreich, für ein embedded System im Einsatz jedoch klarerweise undenkbar.

Deshalb sollen Sie in dieser Übung den Kernel und auch das *Root File System* im Flash des Entwicklungsboards ablegen, sodass das embedded System ohne Netzwerkverbindung starten kann. Dazu sollen Sie sich in diesem Punkt mit dem MTD-Subsystem beschäftigen[1].

Der Linux-Kernel gibt während des Bootens unter anderem folgende Zeilen aus:

```
Creating 3 MTD partitions on "Colibri flash":
0x00000000-0x00080000 : "Bootloader"
0x00080000-0x00480000 : "Kernel"
0x00480000-0x02000000 : "Filesystem"
```

Diese Partitionen sind nicht direkt auf dem Flash angelegt, sondern werden im Kernel in der Datei `drivers/mtd/maps/colibri.c` definiert:

```
static struct mtd_partition colibri_partitions[] = {
    {
        name:           "Bootloader",
        size:           0x00080000, /* 256K U-Boot and
                                       256K config params */
        offset:         0,
    },{
        name:           "Kernel",
        size:           0x00400000, /* 4M for kernel */
        offset:         0x00080000,
    },{
        name:           "Filesystem",
        size:           MTDPART_SIZ_FULL,
        offset:         0x00480000
    }
};
```

[1] Während der Entwicklung von Linux 2.4 wurden die bis dahin existierenden Systeme zur Einbindung von Flashes, der *FTL* (*Flash Translation Layer*) und der *NFTL* (*NAND Flash Translation Layer*), durch das MTD-Subsystem ergänzt. Dadurch konnten nicht nur Patentprobleme umschifft, sondern auch der Zugriff auch solche Speichertypen vereinfacht werden.

Dies bedeutet, dass das Flash in drei Partitionen geteilt wird; diese Partitionen sind auf dem embedded System über die *Device Nodes* `/dev/mtdblock[0-2]` erreichbar. Stellen Sie die Partitionierung des Flashs mit Offset und Größe grafisch dar! □

Meistens werden die Partitionen, in denen der Bootloader (`/dev/mtdblock0`) bzw. der Kernel (`/dev/mtdblock1`) abgelegt werden, ohne Dateisystem ("*raw*") verwendet. Dies vereinfacht das Design des Bootloaders erheblich. Die dritte Partition (`/dev/mtdblock2`) soll als Speicher für das *Root File System* verwendet werden – hier ist also ein Dateisystem (meist ext2/3 oder JFFS2) notwendig.

2. Aufgabe *Kernel-Image und Root File System installieren*

Zuerst sollen Sie den Kernel in Partition 1 (`/dev/mtdblock1`) ablegen. Gehen Sie dazu folgendermaßen vor:

1. Starten Sie das embedded System und booten Sie Linux über das Netzwerk.
2. Beachten Sie die folgende Ausgabe des Bootloaders:

```
## Booting image at 00080000 ...
Bad Magic Number
```

   U-Boot versucht offensichtlich, ein Image von Adresse `0x00080000` zu laden, findet jedoch dort keine passende Signatur (dort ist auch noch kein Image abgelegt!).

3. Nun kopieren Sie das Kernel-Image auf die Flash-Partition 1. Dazu muss es erst für das embedded System erreichbar gemacht werden:

```
host # cp images/uImage rootfs/home/root
```

   Danach ist das Kernel-Image im Dateisystem des embedded Systems in `/home/root` als Datei `uImage` verfügbar. Nun kann es auf das Flash kopiert werden:

```
target # dd if=/home/root/uImage of=/dev/mtdblock1
```

   Achtung: Stellen Sie sicher, dass sie `mtdblock1` verwenden! Wenn Sie das Kernel-Image auf Partition 0 kopieren überschreiben Sie den Bootloader!

4. Wenn Sie nun das embedded System neu starten (`reboot`), sollten Sie bereits einen Unterschied in der Ausgabe des Bootloaders erkennen:

```
## Booting image at 00080000 ...
   Image Name:   Linux Kernel Image
   Created:      2007-10-31  10:17:14 UTC
   Image Type:   ARM Linux Kernel Image (gzip compressed)
   Data Size:    1122890 Bytes =  1.1 MB
   Load Address: a0008000
   Entry Point:  a0008000
   Verifying Checksum ... OK
```

Booten Sie nun das embedded System, ohne es ans Netzwerk angeschlossen zu haben: der Kernel startet aus dem Flash, versucht jedoch weiterhin das *Root File System* über NFS zu mounten. Achten Sie auf die *Kernel Command Line*, die der Kernel beim Booten ausgibt:

```
Kernel command line: root=/dev/nfs ip=:::::eth0: console=ttyS0,9600n8
```

Es sind drei weitere Schritte notwendig, um das *Root File System* im Flash abzulegen: erstens muss ein Dateisystem auf dem Flash erstellt werden, zweitens muss dieses mit den passenden Dateien und Verzeichnissen gefüllt werden, und drittens muss der Kernel dieses Dateisystem als *Root File System* verwenden:

1. Das Dateisystem auf dem Flash wird mit `mkfs.ext2` erstellt. Ermitteln Sie die richtigen Parameter! □

2. Entpacken Sie die Datei `rootfs_min.tbz2` und kopieren Sie den Inhalt auf Ihre neu erstellte Partition. Können Sie dazu `dd` oder `cp`[2] verwenden? Müssen Sie ggf. auf Dateiberechtigungen und/oder den Eigentümer der Dateien achten? Bestimmen Sie dazu eventuell notwendige zusätzliche Kommandozeilenparameter! Geben Sie alle notwendigen Befehle an die Sie zum Entpacken+Kopieren benötigen. □

3. Die *Kernel Command Line* kann an zwei Stellen definiert werden: im Bootloader und als Default-Wert im `.config` des Kernels. Ermitteln Sie, an welcher von diesen beiden Stellen die *Command Line* auf die oben gezeigte eingestellt wird und wie Sie sie manipulieren können. Welchen Parameter müssen Sie für die Option `root=` angeben, um ihr neu erstelltes *Root File System* zu verwenden? Müssen Sie andere Parameter der *Kernel Command Line* ändern? □

Abschließend sollte Ihr System fähig sein, ohne den Entwicklungshost zu starten. Beachten Sie, dass Sie jedoch weiterhin eine Verbindung über die serielle Schnittstelle benötigen, wenn Sie die *Kernel Command Line* im Bootloader änderen müssen!

3. Aufgabe *Hardware-Zugriff*

Schreiben Sie ein Programm, das den aktuellen Zustand der Schalter der Zusatzplatine auf den LEDs ausgibt[3]. Zum Zugriff auf die GPIOs können sie das Kernel-Modul `pxa27x_gpio.ko` verwenden, das Sie über die *Device Node* `/dev/gpio0` steuern: wenn Sie ein Byte auf diese Datei schreiben werden die unteren 4 Bit auf den LEDs ausgegeben, wenn Sie ein Byte davon lesen wird in den unteren 4 Bit der Zustand der Schalter zurückgegeben.

4. Aufgabe *Putting it all together...*

Binden Sie nun Ihr fertiges Programm in das *Root File System* im Flash ein. Vergessen Sie nicht, das verwendete Kernel-Modul auch einzubinden! Überlegen Sie, in welches Verzeichnisse Sie diese Dateien ablegen sollten, um den *Filesystem Hierarchy Standard* zu erfüllen! □

Am Ende dieser Aufgabe soll Ihr System keine Netzwerkverbindung mehr benötigen, um das Programm zu starten!

[2] *Hinweis:* Beim Befehl `cp` können Sie mit der Option `-d` verhindern, dass symbolische Links dereferenziert werden!

[3] Dieses Programm kann die gesamte CPU-Zeit verbrauchen. Sie können also in einer Endlosschleife den Wert der Schalter auf die LEDs schreiben.

5. Aufgabe *Automatischer Start von Applikation+Kernel-Modul*

Offensichtlich ist die bisherige Lösung noch nicht perfekt: das Kernel-Modul sowie die Applikation müssen nach wie vor von Hand geladen werden. Das bedeutet, dass zwar kein Netzwerkanschluss mehr notwendig ist, eine serielle Verbindung jedoch schon.

Konfigurieren Sie Ihr embedded System nun so, dass nach dem Starten des Betriebssystems automatisch das Kernel-Modul geladen und danach ihr Programm gestartet wird: erstellen Sie dazu ein kurzes Script `/etc/init.d/myapp`:

```
1 #!/bin/sh
2 insmod /pxa27x_gpio.ko
3 /home/root/gpio &
```

Beachten Sie, dass Sie das executable-Flag für dieses Script setzen müssen (`chmod` mit passenden Parametern)!

`init` startet beim Booten den *Runlevel* 3. Das bedeutet, dass alle Scripts in `/etc/r3.d/`, deren Name mit `S` beginnt, gestartet werden. Legen Sie also einen symbolischen Link an:

```
1 cd /etc/rc3.d/
2 ln -s ../init.d/myapp S99myapp
```

Hinweis: Natürlich ist die serielle Schnittstelle nach wie vor für den Bootloader notwendig!

"printk("??? No FDIV bug? Lucky you...\n");"
linux-2.2.16/include/asm-i386/bugs.h