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

7. Übung: Windows CE 6.0: Driver Development, Part One

Name(n): Punkte:

1 Windows CE Driver – Stream Interface Device

In Windows CE wird in den meisten Fällen als Schnittstelle zwischen Treiber und Betriebssystem ein so genanntes *Stream Interface* verwendet. Diese Schnittstelle bietet, unabhängig von der zugrundeliegenden Hardware und der Funktionalität des Treibers, nach außen hin immer die selben Funktionen. Typischerweise wird diese Schnittstelle für Geräte verwendet, die Datenströme erzeugen oder verarbeiten wie zB.: UART oder Audiogeräte, aber auch für IO-Geräte wie LEDs oder Schalter. Eine Applikation kann mit dem Treiber interagieren, indem eine spezielle Datei im Filesystem geöffnet wird und auf diese geschrieben bzw. von dieser gelesen wird. Damit der Treiber angesprochen werden kann muss er bzw. dessen Name zuvor vom *Device Manager* registriert werden. Wie schon aus der letzten Übung bekannt ist, wird der Name eines Devices aus einem Präfix (drei Zeichen) und dem Index gebildet, welche der *Registry* entnommen werden.

Die Schnittstelle, welche ein Stream Device zur Verfügung stellt, ist im folgenden aufgelistet.

```
1  DWORD XXX_Init(LPCSTR pContext, DWORD dwBusContext);
2  BOOL  XXX_PreDeinit(DWORD dwContext);
3  BOOL  XXX_Deinit(DWORD dwContext);
4  DWORD XXX_Open(DWORD dwContext, DWORD dwAccess, DWORD dwShare);
5  BOOL  XXX_PreClose(DWORD dwOpen);
6  BOOL  XXX_Close(DWORD dwOpen);
7  DWORD XXX_Read(DWORD dwOpen, LPVOID pBuffer, DWORD dwCount);
8  DWORD XXX_Write(DWORD dwOpen, LPVOID pBuffer, DWORD dwCount);
9  DWORD XXX_Seek(DWORD dwOpen, long lDelta, WORD wType);
10 DWORD XXX_IOControl(DWORD dwOpen, DWORD dwCode, PBYTE pIn, DWORD dwIn,
11                     PBYTE pOut, DWORD dwOut, DWORD *pdwBytesWritten);
12 void XXX_PowerDown(DWORD dwContext);
13 void XXX_PowerUp(DWORD dwContext);
```

Eine Beschreibung der Funktionen und wann diese Aufgerufen werden, ist in [Fun] zu finden.

□ Erklären Sie kurz und eigenen Worten die Bedeutung der einzelnen Funktionen und wann diese Aufgerufen werden.

Das Präfix *XXX*, das jeder Funktion vorangestellt ist, muss durch das entsprechende Kürzel ersetzt werden. Damit der Treiber geladen werden kann, müssen zumindest die Funktionen *Init*, *Deinit*, *Open* und *Close* sowie einer der Funktionen *Read*, *Write*, *Seek* oder *IOControl* vorhanden sein [Bol07].

Der Treiber wird in einer DLL gekapselt, welche beim Systemstart oder zur Laufzeit vom *Device*

Manager geladen wird. Eine DLL, die einen Treiber implementiert, wird vom *Device Manager* immer nur einmal geladen, unabhängig davon ob der Treiber für mehrere Geräte verwendet wird oder mehrere Applikationen auf die selbe Hardware zugreifen. Falls dies benötigt wird, muss sich der Treiber um eine etwaige Synchronisation bzw. Verwaltung der Instanzen kümmern.

2 Gerätetreiber für LEDs, Schalter und Taster

Implementieren Sie den Treiber als DLL und orientieren Sie sich dabei an der Vorlage im Moodle.

2.1 Debugging

Bei der Entwicklung eines Treibers stellt das Debuggen immer ein Problem dar. Eine der einfachsten Möglichkeiten ist die Verwendung von Debug-Ausgaben. Hierfür können Sie folgendes Makro verwenden:

```
RETAILMSG(RETAIL_ON, (TEXT("Ich␣bin␣eine␣Debugausgabe"));
```

Das Define RETAIL_ON ist ein boolscher Ausdruck, der festlegt ob eine Ausgabe erfolgt oder nicht. Die Ausgabe wird auf die serielle Schnittstelle umgeleitet, die die Standard-Schnittstelle für Debug-Ausgaben ist. Dies muss allerdings zuerst im Bootloader eingeschaltet bzw. entsprechend konfiguriert werden. Gehen Sie dabei wie folgt vor [Boo]:

- Verbinden Sie den Host mit der seriellen Schnittstelle des Entwicklungsboards und öffnen Sie ein Terminalprogramm (8N9600).
- Drücken Sie während des Bootens des Colibri-Moduls die Leertaste.
- Drücken Sie *X* um in den *CommandPrompt Mode* zu wechseln.
- Geben Sie folgendes Kommando ein um die serielle Debug-Ausgabe zu aktivieren
  `set dbg.serial=1`
- Um die Änderungen zu speichern müssen Sie anschließend noch folgenden Befehl eingeben:
  `save dbg`

Nun sollten alle Debug-Ausgaben über die serielle Schnittstelle zum Entwicklungshost übertragen werden.

2.2 Implementierung

Die benötigten Hilfsstrukturen sind im Moodle in der Datei *gpio.h* zur Verfügung gestellt. Funktionen für den Zugriff auf die Hardware sind in der Datei *ledswitch.c* bereitgestellt. Erweitern Sie diese Funktionen um den Status der Taster und Schalter auslesen zu können. Informationen bzgl. der entsprechenden GPIO Register entnehmen Sie dem Linux Driver. Beachten Sie bei der Initialisierung der verwendeten PINs, dass zuerst die Funktion und dann die Richtung der PINs eingestellt werden muss.

Um die Einträge in der Registry zu machen, die benötigt werden um den Treiber zur Laufzeit zu laden, können Sie die selben Funktionen wie in Übung 2 verwenden. Das Präfix und der Name der DLL müssen gegebenenfalls angepasst werden.

Implementieren Sie eine Testapplikation, die den Treiber hinreichend testet (auch *Corner-Cases* beachten). □

Beachten Sie bei der Implementierung des Treibers folgende Punkte:

- Beim Entladen des Treibers müssen Sie den Speichers wieder an entsprechender Stelle (Funktion) freigeben.
- Prüfen Sie immer die Rückgabewerte der Funktionen zum Reservieren und Allokieren des Speichers! Sollte ein Fehler auftreten, müssen die bis dahin durchgeführten Funktionsaufrufe wieder rückgängig gemacht werden (in umgekehrter Reihenfolge!) Dazu bietet sich das folgende Konstrukt an[1]:

```
1  if ((err=register_this()))
2      goto fail_this;
3  if ((err=register_that()))
4      goto fail_that;
5  if ((err=register_those()))
6      goto fail_those;
7  ...
8  fail_those:
9      unregister_that();
10 fail_that:
11     unregister_this();
12 fail_this:
13     return err;
```

2.2.1 Speicher mappen

Um auf die GPIOs zugreifen zu können, muss zuerst der entsprechende physikalische Speicher gemappt werden. Dazu haben Sie in der Vorlesung bereits die Funktionen *VirtualAlloc* und *VirtualCopy* kennengelernt. Um den gemappten Speicher anschließend wieder freizugeben, wenn dieser nicht mehr benötigt wird, können Sie die Funktion *VirtualFree* verwenden. Eine entsprechende Vorlage ist in den Dateien *mapreg.h* und *mapreg.cpp* gegeben.

2.2.2 Funktionalität

Der Treiber soll folgende Funktionen beinhalten:

- Es muss nur eine Instanz des Treibers geladen werden können, daher müssen beim Lesen und Schreiben keine Synchronisationsmethoden implementiert werden.

[1] Eine Diskussion über die Sinnhaftigkeit der Verwendung von `goto` finden Sie zB. auf http://kerneltrap.org/node/553/2131. Unabhängig von persönlichen Meinungen über `goto` werden solche Konstrukte im Linux-Kernel oft verwendet und sollten Ihnen daher zumindest geläufig sein.

- Wenn mit der Funktion *WriteFile* Daten geschrieben werden, soll das untere Nibble an den LEDs ausgeben werden. Verwenden Sie dabei folgendes Mapping:

LED	Bit
LED1	Bit3
LED2	Bit2
LED3	Bit1
LED4	Bit0

- Wenn mittels der Funktion *ReadFile* Daten gelesen werden, soll im oberen Nibble der Status der Taster und im unteren Nibble der Status der Schalter enthalten sein. In diesem Fall liegt folgendes Mapping vor:

Schalter/Taster	Bit
Switch0	Bit0
Switch1	Bit1
Switch2	Bit2
Switch3	Bit3
Taster0	Bit4
Taster1	Bit5
Taster2	Bit6
Taster3	Bit7

- Überlegen und kommentieren Sie, zu welchem Zeitpunkt bzw. in welcher Funktion der physikalische Speicher gemappt werden soll. □

References

[Bol07] Douglas Boling. *Programming Windows Embedded CE 6.0*. Microsoft Press Corp., 4th edition, November 2007.

[Boo] Bootloader Menu – Toradex Wiki. Website. Jan 2009, <http://wiki.toradex.com/index.php/Bootloader_Interface>.

[Fun] Stream Interface Device Functions – MSDN Library. Website. May 2007, <http://msdn.microsoft.com/en-us/library/aa930506.aspx>.

[PB08] Stanislav Pavlov and Pavel Belevsky. *Windows Embedded CE 6.0 Fundamentals*. Microsoft Press Corp., 2008.

[WH01] James Y. Wilson and Aspi Havewala. *Building Powerful Platforms with Windows CE*. Addison-Wesley, May 2001.