Dokumentation Shell - Framebuffer

Manuel Jelinek, Martin Erb

30. Januar 2014

Compiler

Damit unser Makefile richtig funktioniert muss folgender Befehl ausgeführt werden welcher den richtigen Compiler für das Raspberry installiert. Der zweit Befehl wird benötigt um raspbootin zu compilieren.

```
sudo apt-get install gcc-arm-linux-gnueabihf
sudo apt-get install g++-arm-linux-gnueabihf
```

Makefile

Die Ordnerstruktur der folgenden Abbildung muss beibehalten werden damit alle targets in den Makefiles ausführbar sind. Um eine Übersicht der verschiedenen targets der Makefiles zu erhalten kann der Befehl `make help` eingegeben werden.
Weiters empfehlen wird unser `raspbootin` zu verwenden, damit auch der Befehl `shutdown` funktioniert.

```
./
├─shell_only/
│   ├─source/
│   │   ├─include/
│   │   │  └─*.h
│   │   ├─init.s
│   │   └─*.c
│   ├─kernel.ld
│   └─Makefile
├─framebuffer/
│   ├─source/
│   │   ├─include/
│   │   │  └─*.h
│   │   ├─init.s
│   │   ├─font.bin
│   │   └─*.c
│   ├─kernel.ld
│   └─Makefile
└─raspbootin/
    ├─raspbootin/
    ├─raspbootcom/
    └─Makefile
```

Shell

Damit die Shell startet und richtig initialisiert wird muss folgende Funktionen aufgerufen werden:

```
shell();
```

Fünf verschiedene Befehle sind in der Shell fix hardcoded implementiert: exit, shutdown, restart, load, help. Was diese Befehle machen kann man nachlesen wenn man die Shell startet und help eingibt. Der Befehl shutdown funktioniert nur wenn auch unser Raspbootcom verwendet wird da wir dort kleine Änderungen vorgenommen haben und somit vom Raspberry aus, Raspbootcom beenden können.
Der nützlichste Befehl ist der load-Befehl. Wenn das Programm einmal auf dem Raspberry ausgeführt wird kann man am PC durch eine zweite Konsole das kernel.img neu erstellen und mit dem Befehle (load) neu auf das Raspberry kopieren und das Programm startet automatisch neu.
Um die Shell möglichst dynamisch zu halten kann man über die Funktion

```
void addNewCommand(fcn_ptr function_pointer, char command_name,
     char* help_text)
```

neue Befehle hinzufügen. Der erste Parameter ist der Pointer auf die Funktion welche beim Aufrufen des Befehls ausgeführt werden soll. Der zwei Parameter ist der Name des Befehls. Der dritte Parameter ist ein char array im welchen der Text steht welche beim aufrufen der Funktion help angezeigt werden soll. Falls der User keinen Hilfe Text anzeigen möchte dann kann muss dieser Pointer auf NULL zeigen. Ein Beispiel dieser Funktion kann man in der shell.c Zeile 21 sehen.

Framebuffer

Initialisieren des Framebuffer am Raspberry Pi: Nur für die Initialisierung des Framebuffers werden folgende Files benötigt:

```
mailbox.(h/c)
memutils.(h/c)
framebuffer.(h/c)
```

Dabei muss nur die File `framebuffer.h` inkludiert werden. Die beiden anderen Files werden zusätzlich vom framebuffer intern verwendet. Um den Framebuffer zu initialisieren, wird entweder die Funktion

```
fbInitNativ();
```

verwendet, die den Framebuffer mit der maximalen Auflösung des Monitors mit einer Farbtiefe von 16bit initialisiert oder man verwendet die Funktion

```
fbInit(size_x, size_y, color_depth);
```

Dieser kann man eine beliebige Auflösung übergeben und als dritten Parameter noch die Farbtiefe. Dabei sind die Farbmodi 16bit, 24bit und 32bit möglich. Für den gewünschten Modus gibt es im Header-File entsprechende Defines:

```
COLORMODE_16BIT
COLORMODE_24BIT
COLORMODE_32BIT
```

Die minimale Auflösung des Framebuffer ist mit 640x480 Pixel und die maximale mit der des Monitors festgelegt. Wird einer der Werte überschritten bzw. unterschritten, so wird der jeweilige Wert auf das entsprechende maximum oder minimum gesetzt. Beispiel:

```
fbInit(1024, 768, COLORMODE_16BIT);
```

Nach der Initialisierung werden Zeichen in weiß auf einem schwarzen Hintergrund dargestellt. Um diese Farben zu ändern, werden folgende Funktionen verwendet:

```
consoleForegroundColor(new_color);  // ändert die Zeichenfarbe
consoleBackgroundColor(new_color);  // ändert die Hintergrundfarbe
```

Einige Standardfarben wurden im Header-File definiert. Dabei sind die Defines vom 32bit Modus ebenfalls für den 24bit Modus zu verwenden (Es wird nur der Alpha-Kanal ignoriert). Beispiele:

```
consoleForegroundColor(COLOR16_RED);
consoleForegroundColor(COLOR32_RED);
consoleBackgroundColor(COLOR16_BLUE);
consoleBackgroundColor(COLOR32_BLUE);
```

Mit der Funktion

```
consoleWriteChar(character)
```

können einzelnen Zeichen in den Framebuffer geschrieben werden. Dabei wird der Standard ASCII Zeichensatz verwendet und um weitere printbare Zeichen der 'CodePage 437' erweitert (Werte > 127). Des weiteren setzt die Funktion die Position, an die das Zeichen geprintet wird auf den nächsten Block weiter bzw. führt am ende der Zeile einen Zeilenumbruch durch. Ist das Ende des Framebuffers erreicht, so werden alle Zeichen um eine Zeile nach oben verschoben und die letzte Zeile gelöscht. Mit der Funktion

```
clearScreen()
```

kann der Framebuffer wieder gelöscht werden. Des weiters wird der Cursor wieder an die linke obere Position des Monitors gesetzt.