

nucleOS

Konzept v1.0

by MrSaint & rochus

14. Dezember 2003

Inhaltsverzeichnis

1	Übersicht	3
1.1	Die Compiler	3
2	Allgemeines	3
2.1	Der Bootvorgang	3
2.2	Kernel, Shell und Treiber	4
3	Der Bootloader	4
3.1	Eigenschaften.....	4
3.2	Roadmap	4
4	Der Kernel-Wrapper	4
4.1	Eigenschaften.....	4
4.2	Beschreibung	4
5	Der Kernel.....	5
5.1	Unterteilung in Gruppen	5
5.2	Eigenschaften.....	5
5.3	Roadmap	5
5.4	Beschreibung	5
6	Treiber	6
7	Sonstige, wichtige Programme	6
7.1	Shell	6
7.2	mount	6
7.3	dir	6
7.4	cd	6

1 Übersicht

Die Entwicklung des Betriebssystems (im Folgenden „OS“ für „Operating System“ genannt) umfasst 4 wesentliche Teile: Die Entwicklung

- des Bootloaders
- des Kernel-Wrappers
- des Kernels
- der Treiber
- sonstiger, wichtiger Programme.

Einzelheiten werden hier besprochen.

1.1 Die Compiler

Als Compiler sind vorgesehen:

- der Netwide Assembler oder kurz NASM (<http://nasm.sourceforge.net>).

Der Pascal Compiler steht noch nicht endgültig fest, da es Probleme bei der Generierung von flat-binary Dateien gibt. Zur Auswahl stehen im Moment:

- FreePascal (<http://www.freepascal.org>) und
- GNU Pascal (<http://www.gnu-pascal.de>)

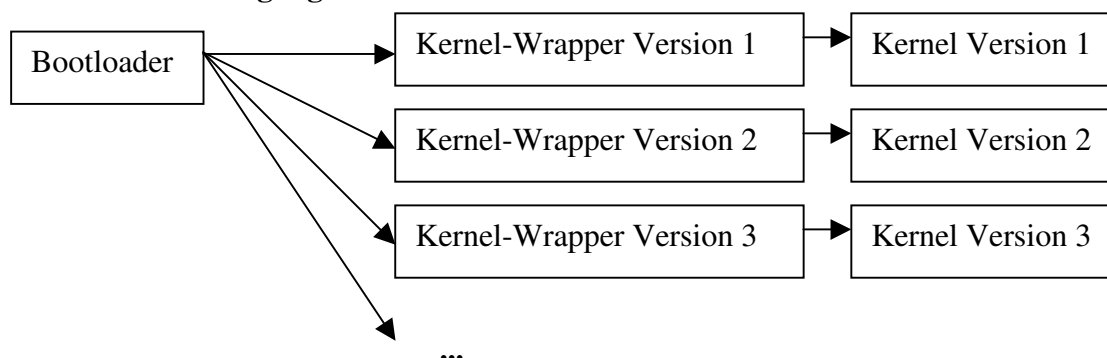
2 Allgemeines

Die Entwickler müssen sich zum Anfang in bestimmte Gruppen aufteilen, die je den gleichen Anteil an Programmierarbeit bekommen. So sollte eine Gruppe aus Programmierern, die Assembler (im Folgenden kurz ASM genannt) programmieren können, die Entwicklung des Bootloaders übernehmen etc. An der Entwicklung des Kernels, des Kernel-Wrappers und der Treiber sollten allerdings alle Mitarbeiter mitarbeiten, da dies die wichtigsten und umfangreichsten Punkte sind.

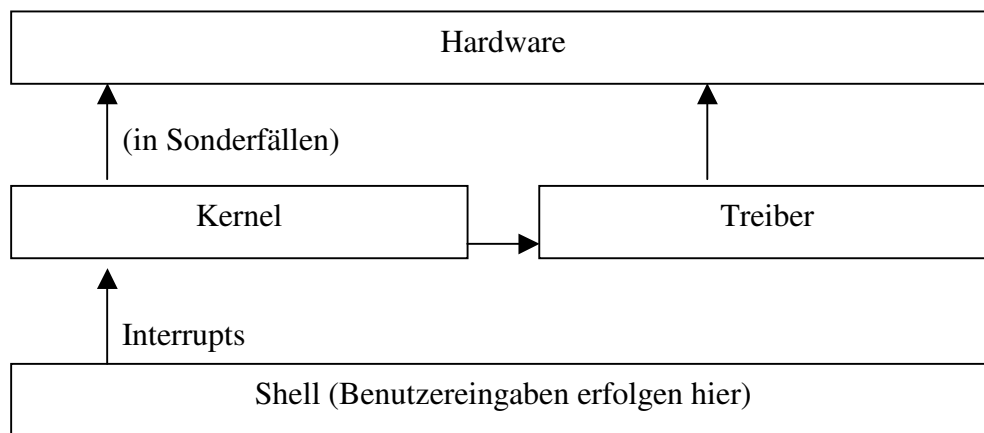
Das OS soll ein 32bit OS sein. Vielleicht wird es später auf 64bit erweitert.

In den folgenden Unterpunkten ist die Situation, wie sie am Ende der Programmierung aussehen sollte, grafisch dargestellt.

2.1 Der Bootvorgang



2.2 Kernel, Shell und Treiber



3 Der Bootloader

3.1 Eigenschaften

- komplett in Assembler (im Folgenden kurz ASM genannt) geschrieben
- komplett textbasiert

3.2 Roadmap

1. Programm schreiben, das sich booten lässt
2. Textausgabe, was gerade passiert
3. Laden eines Programms (späterer Kernel) in den RAM (Zusammenarbeit mit Kernel-Dateisystem-Gruppe)
4. Im RAM befindliches Programm ausführen
5. Wenn verschiedene Kernel vorhanden sind, soll ausgewählt werden können, welcher gebootet werden soll (möglichst mit Versionsangabe)

4 Der Kernel-Wrapper

4.1 Eigenschaften

- komplett in ASM geschrieben
- regelt, dass der Kernel geladen werden kann

4.2 Beschreibung

Da wir den Kernel (im Moment) nicht als flat-binary schreiben können, müssen wir dafür sorgen, dass der Kernel in der richtigen Form in den RAM geschrieben wird und von da aus problemlos ausgeführt werden kann. Dies macht der Kernel-Wrapper.

5 Der Kernel

5.1 Unterteilung in Gruppen

Die Kernel-Gruppe sollte sich noch einmal in verschiedene Untergruppen aufteilen:

- Speichermanagement Gruppe
- Dateisystem Gruppe
- Interrupt Gruppe

5.2 Eigenschaften

- in Pascal geschrieben
- Ansprechen des Kernels von Außerhalb mit Interrupts (wie die DOS-Interrupts) (→ PIC, IDT)
- Strenge Trennung zwischen Kernel-Mode und User-Mode
- User-Accounts (wie bei UNIX; Superuser=„root“)
- Benennung von Festplatten wie bei UNIX (mit „/“ als Root -Verzeichnis, darin werden Partitionen gemountet → 6.2)

5.3 Roadmap

1. Einfacher Kernel mit Textausgabe
2. Laden von Dateien in den RAM und Ausführen dieser
3. Implementation eines Dateisystems
4. Implementation Protected Mode
5. Implementation Speichermanagement
6. Implementation Interrupts
 - a. Interrupt für Ein- und Ausgabe von Zeichen
 - b. ...
 - c. Interrupt zum Ausführen von Programmen
7. Ansprechen anderer Hardware und Peripherie
 - a. Netzwerk
 - b. DMA
 - c. Grafik
 - d. ...
8. Multitasking

5.4 Beschreibung

Von Anfang an muss beachtet werden, dass der Kernel zu gegebener Zeit zu einem Multitasking-Kernel ausgebaut werden soll.

Der Kernel soll ein sogenannter Microkernel werden. Wenn also eine Anfrage von einem Programm kommt, z.B. eine Datei zu öffnen, leitet der Kernel die Anfrage an den dafür zuständigen Treiber, in diesem Fall an den Dateisystem-Treiber weiter. Dies hält den Kernel flexibel und möglichst klein.

Trotzdem ist es nötig, einen Festplatten- (oder Floppy-) Treiber und einen Dateisystem-Treiber direkt in den Kernel zu implementieren, da sich die anderen Treiber ja auf Festplatte (oder Floppy) befinden und man diese ja lesen muss, sonst kann man sie nicht in den RAM schreiben und auch nicht auf sie zugreifen.

6 Treiber

Es werden für das OS auch Treiber benötigt, die das OS ansprechen kann. Sie sollten auch „verschachtelbar“ sein, da z.B. ein Dateisystem -Treiber den Festplatten-Treiber benötigt, um die angeforderten Operationen durchzuführen.

Treiber müssen aus flat-binary-Dateien bestehen. Der Code dieser muss mit dem ersten Zeichen beginnen (das erste Zeichen muss ein Opcode sein). Es wird festgelegt, dass sich in den CPU-Registern EAX:EBX beim Aufruf des Treibers eine Adresse befindet, die auf Parameter, die dem Treiber übergeben werden müssen, zeigt. Die Parameter sind frei wählbar, es muss allerdings eine gemeinsame Grundstruktur gewählt werden. Das Trennzeichen für Parameter im Speicher ist FFh, das Ende der Parameterliste wird durch 00h angezeigt.

7 Sonstige, wichtige Programme

7.1 Shell

Die Shell ist das Programm, welches Programme startet, Verzeichnisse wechselt etc. Sie wendet praktisch nur die vom Kernel bereitgestellten Interrupts an.

7.2 mount

Mountet einzelne Partitionen in bestimmte Verzeichnisse. Die Funktion ist dem UNIX-mount-Befehl gleich.

7.3 dir

Zeigt den Ordnerinhalt des aktuellen Ordners an.

7.4 cd

Wechselt das aktuelle Arbeitsverzeichnis („change directory“)