SysProg Filesystem

Im Rahmen der abschließenden drei Übungsaufgaben sollen Sie ein einfaches Dateisystem implementieren. Wir stellen Ihnen dazu Funktionen in Form von Objekt- und Headerdateien zur Verfügung, die einen Disktreiber für eine physikalische Disk simulieren. Tatsächlich legen diese Funktionen unter /usr/tmp eine Datei an, die Ihr Filesystem repräsentiert. Die Diskfunktionen arbeiten auf 512 Byte großen Diskblöcken.

- **int new_disk(unsigned long size, char *name):**
  Legt unter /usr/tmp eine neues Diskimage unter dem Namen SPFS<name>, mit der angegebenen Länge an. Alle weiteren Funktionen arbeiten dann auf dieser Datei. Im Fehlerfall liefert die Funktion -1, sonst 0.
- **int find_disk(char *name):**
  Wie new_disk(), jedoch wird keine neue Datei angelegt, sondern mit der angegebenen Datei gearbeitet.
- **int write_block(block_t block, unsigned long blocknr):**
  Schreibt den übergebenen Block an die durch *blocknr* angegebene Stelle und liefert die Anzahl geschriebener Bytes, bzw. -1 im Fehlerfall zurück.
- **int read_block(block_t block, unsigned long blocknr):**
  Liest einen Diskblock an der Stelle *blocknr* in den übergebenen Block *block*.
- **void panic(char *message):**
  Dient zur Ausgabe einer Fehlermeldung und dem Abbrechen des Programms und sollte nur bei nicht korrigierbaren Fehlern aufgerufen werden.

Unter dem Verzeichnis /proj/i4sp/pub/aufgabe11 finden Sie sowohl die erwähnte Headerdatei diskdriver.h , als auch die Objektdatei diskdriver.o, in der die oben erwähnten Funktionen enthalten sind.

Das zu implementierende Dateisystem ähnelt dem System V Dateisystem. Die Disk ist dabei folgendermaßen aufzuteilen:

Bootblock	Superblock	Inodes	Datenblocks
0	1		

- **Bootblock**: Dieser Block (Blocknummer 0) enthält die Information, die zum Booten eines Systems notwendig ist. Ihr Dateisystem soll diesen Block reservieren.
- **Superblock**: Dieser Block enthält die folgende Information (siehe block_intern.h)
  - Die Größe der Disk in Blöcken
  - Die Anzahl der Blocks, die für die Speicherung der Inodes verwendet werden
  - Die Anzahl freier Inodes und Datenblöcke
  - Eine beschränkte Liste mit den Nummern freier Inodes.
  - Die Nummer des ersten Datenblocks zur Speicherung freier Datenblocknummern.
- **Inodes**: Blöcke, die die Daten aller Inodes des Dateisystems enthalten
- **Datenblocks**: Die restlichen Datenblocks

Inodes

Das spfs kennt ähnlich dem System V Filesystem Inodes. Diese enthalten in unserem Fall die folgenden Informationen:

```
struct dinode {
    u16_t di_mode;
    u16_t di_nlinks;
    u32_t di_size;
    u32_t di_addr[MAX_BLOCKS_PER_FILE]; /* 108 Byte / 4 */
    u32_t di_atime;
    u32_t di_mtime;
    u32_t di_ctime;
};
```

Die Felder haben dabei folgende Bedeutung (ähnlich wie bei stat):

- **di_mode**: Typ der Inode (Regular File, Directory, Free) und Zugriffsrechte
- **di_nlinks**: Anzahl der Links, d.h. Directoryeinträge für die Datei
- **di_size**: Dateigröße in Bytes
- **di_addr**: Maximal 27 Blocknummern, aus denen die eigentliche Datei besteht
- **di_atime, di_mtime, di_ctime**: Zugriffszeiten auf die Datei

Der Superblock enthält eine Liste mit freien Inodes. Wird eine Inode angefordert, wird das letzte Element dieser Liste entfernt. Sollte diese Liste leer sein, wird in den Inodeblöcken nach freien Inodes (di_mode == 0) gesucht und die Liste wieder aufgefüllt.

Wird eine Inode freigegeben, wird sie hinten angefügt, solange in der Liste noch Platz ist. Sonst wird die Inode nur als frei gekennzeichnet (di_mode == 0).

Maximal soll das spfs **256** Inodes unterstützen!

Blockverwaltung

Der Superblock vom spfs enthält im Unterschied zum System V Filesystem selbst keine Referenzen auf freie Blöcke, sondern nur die Nummer des ersten Blockes, der für Verwaltung freier Blökke verwendet wird. Dieser enthält Nummern freier Blöcke und ggf. im ersten Feld eine Referenz auf den nächsten Verwaltungsblock.

Bei einer Blockanforderung wird immer aus dem ersten Verwaltungsblock die Nummer eines freien Blockes ermittelt. Enthält dieser Verwaltungsblock nur noch die Referenz auf den nächsten Verwaltungsblock, wird er selbst als Block zurückgegeben. Im Superblock wird dann auf den nachfolgenden Verwaltungsblock verwiesen.

Achtung:

Denken Sie daran, vor dem Verlassen Ihres Arbeitsplatzes das Diskimage Ihres Dateisystems unter /usr/tmp wieder zu löschen!

Aufgabe 11:

spfs - Diskstruktur und Inodeverwaltung (15 Punkte)

In der Datei /proj/i4sp/pub/aufgabe11/src/block.h finden Sie eine Headerdatei mit den wichtigsten Deklarationen und den Prototypen von Funktionen, die Sie zu implementieren haben:

a) int print_superblock()

Diese Funktion gibt die im Superblock enthaltene Information auf den Fehlerausgabekanal aus und soll zu Debuggingzwecken dienen. Die Formattierung der Information bleibt Ihnen überlassen.

b) u32_t get_n_free_blocks()

Diese Funktion gibt die Zahl freier Datenblöcke auf Ihrem Dateisystem zurück.

c) u32_t get_n_free_inodes()

Diese Funktion gibt die Zahl freier Inodes Ihres Dateisystems zurück.

d) u32_t get_disksize()

Die Funktion gibt die Gesamtgröße Ihres Dateisystems in Blöcken (siehe block.h) zurück.

e) int sp_format(u32_t disksize)

Diese Funktion formattiert ein Diskimage der Größe *disksize*. Dazu gehören die folgenden Schritte:

- Bootblock initialisieren: Ihr Dateisystem soll den Block 0 als Bootblock reservieren und mit Null initialisieren.
- Superblock erzeugen: Ein Superblock ist zu erzeugen. In den Superblock eingetragen werden die notwendigen Informationen über die Diskgröße, die Anzahl der Inodeblocks, die Anzahl freier Inodes und Datenblöcke, sowie die Informationen über freie Inodes und Datenblöcke.
- Inodes erzeugen: Legen Sie 256 Inodes an und initialisieren Sie diese.
- Blockverwaltung: Verketten Sie die freien Datenblöcke mit dem vorhin beschriebenen Verfahren

f) inoNo_t allocate_inode()

Liefert die Nummer einer freien Inode zurück.

g) int reclaim_inode(inoNo_t inodenr)

Gibt die angegebene Inode wieder frei.

h) int write_inode(inoNo_t inodenr, struct dinode_t *inode)

Schreibt die Inodeinformation aus *inode* auf die Platteinode, die durch *inodenr* bezeichnet ist.

i) int read_inode(inoNo_t inodenr, struct dinode *inode)

Liest die durch *inodenr* festgelegte Platteninode in die Inodestruktur ein, die durch *inode* referenziert wird.

Zum Testen Ihrer Formattierung und der beschriebenen Funktionen stehen Ihnen Testprogramme unter /proj/i4sp/pub/aufgabe11/bin.xxx zur Verfügung. Dasselbe gilt für die notwendigen Objektdateien, die Sie zur Lösung benötigen.

Abgabetermin: 06.02.98 bis 12.00 Uhr