

Musterlösungen zu den Tutoraufgaben auf
Blatt 11
der Übungen zur Vorlesung
“Grundlagen Betriebssysteme und Systemsoftware

G.Groh, 21.01.2009

Aufgabe 4.1 Lösungsvorschlag

Anforderungen an Dateisysteme:

- eine große Menge an Informationen muss gespeichert werden können
- die Informationen müssen persistent (=dauerhaft) gespeichert werden; sie müssen also die Terminierung (=Beendigung) der Prozesse, die sie erstellt haben, überleben
- mehrere Prozesse müssen gleichzeitig auf die Informationen zugreifen können
- das Auffinden gespeicherter Informationen muss effizient möglich sein

Fast alle Dateisysteme sind hierarchisch organisiert, wobei mehrere Dateinamen in eine Gruppe zusammengefasst werden, die Verzeichnis genannt wird.

Aufgabe 4.1 Lösungsvorschlag

Beispiele für Dateisysteme:

- Viele frühe Betriebssysteme (z.B. CP/M, Apple DOS, Commodore DOS) hatten jeweils nur ein Dateisystem, welches daher keinen eigenen Namen trug.
- Neuere Betriebssysteme:
 1. Linux/UNIX:
 - * minix
 - * ext2
 - * ext3 (Variante von ext2 mit journaling)
 - * ReiserFS (Linux Journaling File System von Hans Reiser)
 - * JFS (Journaled File System)
 - * UFS (UNIX File System, verwendet unter Solaris)
 - * XFS Journaling-Dateisystem
 - * xFS Netzwerk-Dateisystem
 - * NFS Network File System (von Sun für Solaris entwickelt)

Aufgabe 4.1 Lösungsvorschlag

Beispiele für Dateisysteme:

2. MS-DOS:

- * FAT bzw. FAT12 (File Allocation Table, für Disketten)
- * FAT16 (Erweitertes FAT-System für Festplatten)

3. MS-Windows:

- * FAT32 (Erweitertes FAT für große Festplatten)
- * VFAT (Erweiterung zu allen FAT-Systemen, längere Dateinamen)
- * NTFS

4. AmigaOS:

- * FFS (Amiga Fast File System)

Aufgabe 4.1 Lösungsvorschlag

Beispiele für Dateisysteme:

5. Apple Macintosh:

- * ProDOS (Dateisystem der späten Apple II-Modelle)
- * MFS (Macintosh File System)]
- * HFS (Hierarchical File System)
- * HFS+ (Erweiterung von HFS u.a. auf Dateinamen mit mehr als 32 Zeichen)

Aufgabe 4.1 Lösungsvorschlag

Beispiel: Reiser FS

Das ReiserFS ist ein Mehrzweckdateisystem. Es wurde von einer Gruppe um Hans Reiser entwickelt und realisiert. Zur Zeit wird es nur von Linux unterstützt. Unter Linux (Version 2.4.1) war ReiserFS das erste Journaling-Dateisystem, welches standardmäßig im Linux Kernel implementiert war.

Der Nachteil von ReiserFS ist, dass das Journaling nur für die Metainformationen, d.h. für die Verzeichnisse und Verwaltungssektoren, nicht jedoch für die Nutzdaten in den Dateien selbst angewendet wird.

Vorteile, gegenüber anderen Dateisystemen, bietet ReiserFS vor allem bei der Handhabung von vielen kleinen Dateien, da diese in den Verwaltungsknoten gespeichert werden können.

Das ReiserFS verwendet eine B*-tree-Struktur um die Daten zu verwalten.

Aufgabe 5 Lösungsvorschlag

1. Realisierungsentscheidungen in Unix

(a) Eine Unix-Datei ist eine **Folge von Bytes**, ohne weitere vom Unix-Kern definierte Struktur (alternativ dazu gibt es noch ein Folge von Rekords (CP/M) oder einen sortierten Baum). Es werden verschiedene Dateiarten in Unix unterschieden:

- **ordinary** bzw. **regular Files** sind "normale" Dateien
- **directories** enthalten Verweise auf andere Dateien und directories; ermöglichen die Strukturierung des Datei-Systems
- **character special** und **block special Files** sind Gerätedateien für zeichen- und blockorientierte I/O-Geräte
- **pipes** sind Kommunikationspfade mit FIFO-Semantik zwischen zwei Prozessen

Aufgabe 5 Lösungsvorschlag

(b) Es gibt folgende Systemdienste (system calls) im Zusammenhang mit dem Dateisystem:

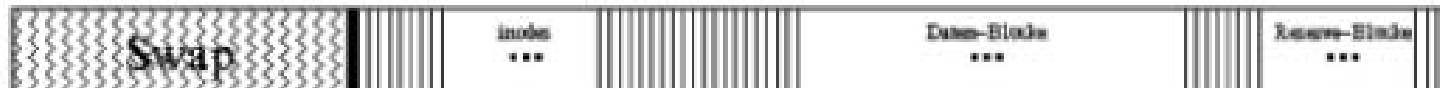
- `file_descr := create(pathname,mode)`: Erzeugen einer neuen Datei
- `file_descr := open(pathname,mode)`: Öffnen einer Datei zum Lesen und/oder Schreiben
- `status := close(file_descriptor)`: Schließen einer geöffneten Datei
- `number := read(file_descriptor,buffer,number_bytes)`: Lesen einer Datei
- `number := write(file_descriptor,buffer,number_bytes)`: Schreiben einer Datei
- `pos := lseek(file_descriptor,offset,reference)`: Positionieren des Datei-Zeigers
- `status := stat(pathname,statbuffer)`: liefert Informationen über Größe, Besitzer, Rechte, etc. einer Datei
- `status := mkdir(pathname,mode)`: Erzeugen eines neues Directory
- `status := rmdir(pathname)`: Löschen eines leeren Directory
- `status := chdir(pathname)`: Wechseln des Working-Directory
- `status := chown(pathname,owner,group)`: Ändern des Besitzers, der Gruppe einer Datei

Aufgabe 5 Lösungsvorschlag

- `status := chmod(pathname,mode)`: Ändern der Rechte an einer Datei
- `status := mount(...)`: "Einhängen" (mounten) eines Dateisystems in ein Directory eines anderen Dateisystems; entsprechend `umount`
- `status := link(path1,path2)`: erzeugt einen neuen Directory-Eintrag `path2` auf eine bereits existierende Datei `path1`
- `status := unlink(path)`: Löschen eines Directory-Eintrags für eine Datei; wenn der letzte link auf eine Datei gelöscht ist, wird die Datei aufgelöst.
- `status := mmap(...)`: Blende eine Datei in den virtuellen Adressraum ein; entsprechend `munmap`. Hierfür werden die entsprechenden Blöcke als Hintergrundspeicher für die Seiten verwendet. Somit können die Standardmechanismen der Speicherverwaltung verwendet werden. So werden Änderungen etwa beim Verdrängen einer Seite in die Datei geschrieben. Bei Terminierung eines Prozesses werden alle eingeblendeten, modifizierten Seiten zurück in die Datei geschrieben.

Aufgabe 5 Lösungsvorschlag

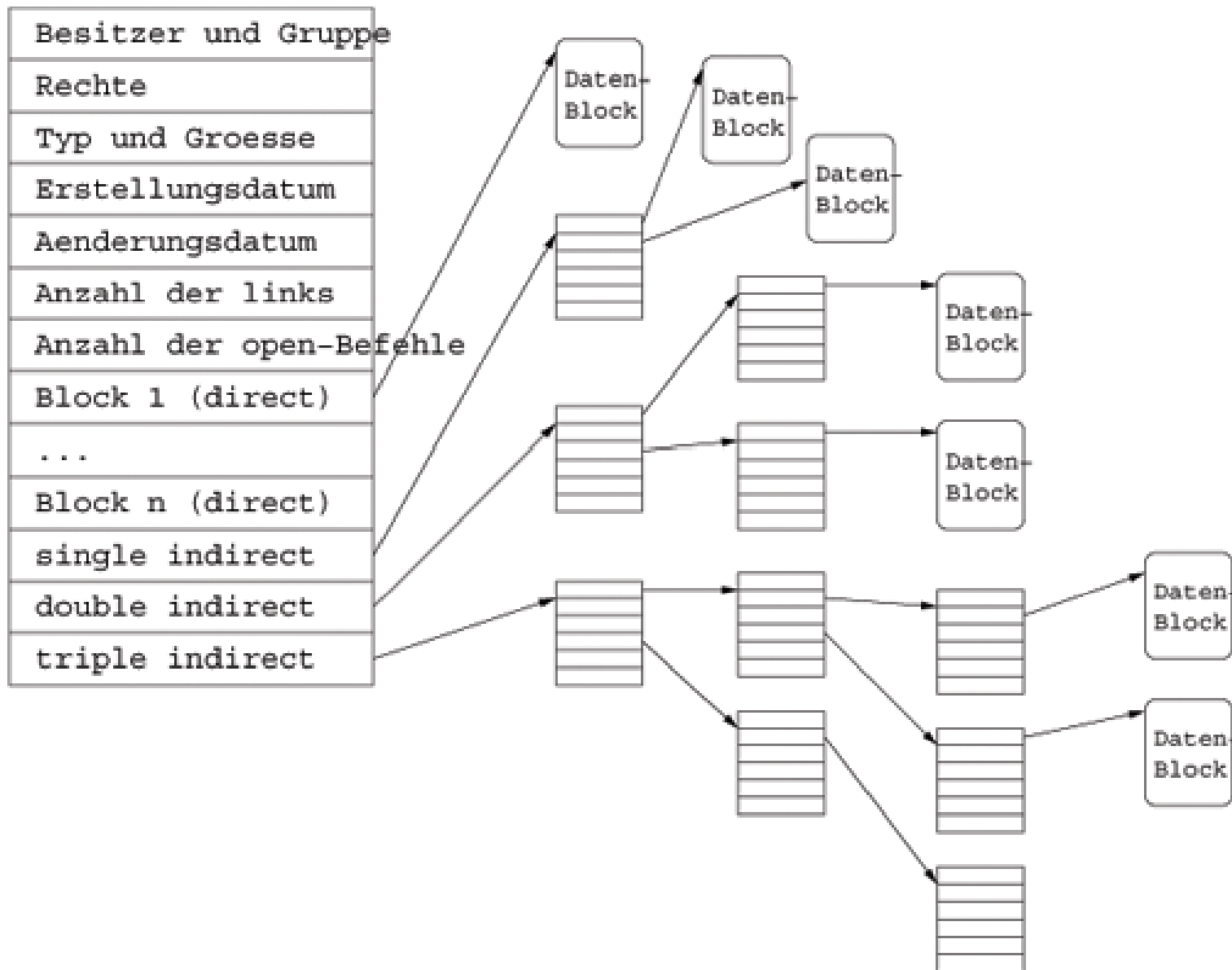
- (c) Das Unix-Dateisystem ist hierarchisch strukturiert und zwar baumartig mit dem "root"-Directory als Wurzel. Die strenge Baum-Struktur wird durch Links aufgebrochen, so daß aus dem Baum i.d.R. ein azyklischer Graph wird.
2. Der Hintergrundspeicher wird in Blöcke unterteilt. Dabei wird ein Teil der Blöcke verwendet, um die Belegt-Liste der Datenblöcke zu realisieren. In Unix werden außerdem einige Blöcke als Reserve gehalten, die im Falle des defekt werdens zum Einsatz kommen, so dass die Kapazität der Platte trotz defekter Blöcke nicht kleiner wird. Auf die Datenblöcke kann nur über die `inodes` zugegriffen werden. In Unix-Systemen wird außerdem auf der Festplatte eine Swap-Partition zum Auslagern von Seiten angelegt.



Aufgabe 5 Lösungsvorschlag

- (a) Die Belegt-Liste wird mit Hilfe von "inodes" realisiert. Jede Datei wird mit genau einem inode beschrieben. Es wird der Name, der Benutzer und die Gruppe, die Rechte, der Typ, die Größe und Erstellungs- und Änderungsdatum festgehalten. Die Blöcke, die zur Realisierung der Datei verwendet wurden, werden in eine Liste im inode direkt eingetragen (Größenordnung etwa 10 bis 1000 Stück, je nach inode-Größe). Werden mehr Blöcke benötigt, so verweist ein Eintrag im inode auf einen zusätzlichen inode, in dem sich nur Verweise auf Blöcke befinden (single indirekt). Im weiteren gibt es noch einen double und einen tripple indirekt Eintrag. So können kleine Dateien sehr effizient direkt adressiert werden. Die dreifache Indirektion bietet im Gegensatz dazu die Möglichkeit mehrere Millionen Blöcke anzusprechen.

Aufgabe 5 Lösungsvorschlag



Aufgabe 5 Lösungsvorschlag

- (b) Verzeichnisse (directories) werden in Unix wie Dateien mit Hilfe eines inodes angelegt. Der inode verweist auf Datenblöcke, die in Tabellenform Dateinamen (bzw. Unterverzeichnisnamen) und die zugehörige inode-Nummer enthalten.
- (c) Ein Link ist ein Directory-Eintrag, der auf eine bereits existierende Datei verweist, d.h. eine Datei kann unter verschiedenen Namen in verschiedenen Directories referenziert werden. Die entsprechenden Directory-Einträge zeigen jedoch alle auf den gleichen inode. Von diesen sogenannten hard links sind Soft-Links (oder auch symbolic links) zu unterscheiden. Soft-Links sind Dateien, die einen Pfadnamen enthalten, und so auf eine andere Datei oder ein Directory zeigen.

Mit dem Link-Konzept läßt sich die strenge hierarchische Baumstruktur aufweichen. Vorteile sind u.a. kürzere Pfadnamen, um von einem Zweig des Baumes in einen anderen zu gelangen, ist es nicht mehr notwendig den absoluten Pfadnamen über die Wurzel anzugeben. Bei Hard-Links kommt hinzu, daß die mehrfache Belegung von Plattenspeicherplatz vermieden wird, da keine unnötigen Kopien von Dateien erzeugt werden müssen. Nachteilig ist der zusätzlich erforderliche Verwaltungsaufwand, der wesentlich darin besteht, daß man sich für jede Datei, die Anzahl der Links, die es auf diese Datei gibt, merken muß. Die Datei darf erst dann gelöscht werden, wenn keine Links auf diese Datei mehr existieren.

Aufgabe 5 Lösungsvorschlag

3. (a) In Unix lassen sich die Rechtefestlegungen für eine Datei über einen neun Bit-Vektor angeben. An einer regulären Datei lassen sich die Rechte zur Ausführung der Operationen read (r), write (w) und execute (x) vergeben und zwar für den Eigentümer (User) der Datei, die Mitglieder der eingetragenen Gruppe (Group) und alle anderen (Others). Für Directories entspricht das execute-Recht dem Recht zum Wechseln in das Directory. Die ersten drei Bits legen die Rechte für den User fest, die nächsten drei Bits die Rechte der Gruppenmitglieder der Gruppe des Users und die letzten drei Bits die Rechte aller anderen Benutzer.

Ein kleines Beispiel: `rw- r-- ---` bedeutet: der User (Eigentümer) der Datei kann die Datei lesen und schreiben, andere Mitglieder der Gruppe des Users können die Datei nur lesen und alle anderen Benutzer haben keine Rechte an der Datei. Die Rechtefestlegungen für eine Datei lassen sich von dem Eigentümer der Datei mit Hilfe des Kommandos `chmod` (siehe oben) ändern.

Aufgabe 5 Lösungsvorschlag

- (b) Zugriffskontrollen werden in Unix folgendermaßen durchgeführt: Zur Beschreibung eines Prozesses gehört die uid des Benutzers, von dem er gestartet wurde, und die gid der Gruppe dieses Benutzers. Wenn ein Prozess nun eine Datei öffnen möchte (siehe system call `open`), so muss er angeben, welche Operationen er auf der Datei ausführen möchte. Der Unix-Kern überprüft dann anhand der in dem inode der Datei enthaltenen uid/gid und Rechtfestlegungen sowie der uid/gid des Prozesses, ob der Prozess die Rechte zur Ausführung dieser Operationen hat. Ist dies der Fall, so erhält der Prozess einen File-Deskriptor, der ihm die entsprechende Nutzung der Datei erlaubt, zurück. In dem entsprechenden Eintrag in der Open-File-Table des Systems wird vermerkt, ob die Datei zum Lesen und/oder Schreiben geöffnet ist. Damit kann bei jedem Zugriff auf die geöffnete Datei überprüft werden, ob der Zugriff gemäß der beim Open angegebenen Zugriffsoperationen auch erlaubt ist.

Aufgabe 5 Lösungsvorschlag

- (c) Mängel und Schwachstellen des Unix-Schutzkonzepts u.a.: Außer für den Eigentümer lassen sich nicht für einzelne Benutzer Rechtesfestlegungen individuell treffen. Die Rechtesfestlegungen sind also nur sehr grob möglich. Ein Benutzer kann nicht speziell für einzelne weitere Benutzer Rechte an seinen Dateien vergeben.
- Rechtesänderungen, insbesondere Rechtesrücknahmen, an einer Datei betreffen nicht die Prozesse, die diese Datei in geöffnetem Zustand halten. Hat also ein Benutzer das Schreibrecht an einer Datei und hat er die Datei geöffnet, so kann er, falls das Schreibrecht für ihn zurückgenommen wird, weiter solange auf die Datei schreiben, wie er die Datei geöffnet hält. Der Mangel liegt darin begründet, daß in Unix nicht bei jedem Zugriff auf eine Datei die Berechtigung überprüft wird, sondern nur genau einmal, nämlich beim Öffnen der Datei.