Betriebssysteme

Tutorium 6

Philipp Kirchhofer
philipp.kirchhofer@student.kit.edu
http://www.stud.uni-karlsruhe.de/~uxbtt/

Lehrstuhl Systemarchitektur
Universität Karlsruhe (TH)

20. Januar 2010

Was machen wir heute?

1. Tutorien Übungsblatt
   - File System Basics
   - Implementing Random Access to Files
   - Virtual File System
   - Fun with Files in Linux

Tutorien Übungsblatt - File System Basics

Frage 10.1.a

Welche zwei Zugriffsmethoden auf eine Datei gibt es?

Antwort

Sequentieller Zugriff
Die Datei wird nur sequentiell durchlaufen, es gibt keine Sprünge. Eine Leseoperation liest eine bestimmte Anzahl an Bytes ein und setzt die Position innerhalb der Datei um die entsprechende Länge weiter.

Wahlfreier Zugriff
Programme können beliebig Teile der Datei lesen oder beschreiben. Im Gegensatz zum sequentiellen Zugriff muss der Anwendungsprogrammierer den genauen Zugriffsort angeben.

Tutorien Übungsblatt - File System Basics

Frage 10.1.b

Bei manchen Betriebssystemen können Anwendungen direkt über die Auswahl einer Datendatei gestartet werden. Wie kann diese Zuordnung zwischen Datendatei und Anwendung umgesetzt werden?

Antwort

- Die Anwendung kann in der Datei selber vermerkt werden
- Dateiendung
- Dateityperkennung („magic“)

Frage 10.1.c

Was ist der Unterschied zwischen einem relativen und einem absoluten Dateinamen?

Antwort

Ein absoluter Dateiname startet am obersten Verzeichnis im Dateisystembaum, ein relativer Dateiname dagegen wird relativ zu dem aktuellen Verzeichnis interpretiert.

Beispiel: Absolute und relative Dateinamen

Aktuelles Verzeichnis: */tmp/pages/kernel/*
Beide Verweise zeigen auf die gleiche Datei.

Absolut
/tmp/files/sysarch

Relativ
../../files/sysarch

Frage 10.1.d

Was ist der Unterschied zwischen einem symbolischen Link und einem Hardlink?

Antwort

Ein symbolischer Link ist eine spezielle Datei, die einen absoluten oder relativen Pfad zu der referenzierten Datei enthält.

Ein Hardlink dagegen ist ein Verzeichniseintrag, der auf den Metadateneintrag (Inode) der referenzierten Datei verweist. Hardlinks unterscheiden sich in der weiteren Handhabung nicht mehr von der referenzierten Datei. Bei Hardlinks müssen der Hardlink und die referenzierte Datei im gleichen Dateisystem liegen.

Frage 10.1.e

Was ist eine Zugriffskontrollliste (access control list, ACL)? Welche Probleme können mit ACLs auftreten und wie können diese Probleme gelöst werden?

Antwort

Eine ACL ist eine Liste, die mit einer Datei verknüpft ist. Die Liste enthält einen oder mehrere Einträge der Form: „benutzername: zugriffsrecht“. Eine ACL erlaubt eine feingranularen Kontrolle darüber, welcher Benutzer welche Operationen auf der Datei ausführen darf.

Ein Problem ist die fehlende Skalierbarkeit. Wenn ein neuer Benutzer in einem System hinzugefügt wird muss in jeder Datei, die von dem Benutzer gelesen werden darf, ein neuer ACL Eintrag hinzugefügt werden. Die ACL Listen werden bei vielen Benutzern zudem unhandlich groß.

Eine Lösung ist der Einsatz von Gruppen. In jeweils einer gleichen Gruppen sind Benutzer zusammengefasst, die die gleichen Operationen auf Dateien durchführen dürfen. Ein Benutzer kann Mitglied in mehreren Gruppen sein.

Frage 10.1.f

Welche Datenstrukturen werden im Betriebssystemkern für ein Unix ähnliches Dateimanagement benötigt?

Antwort

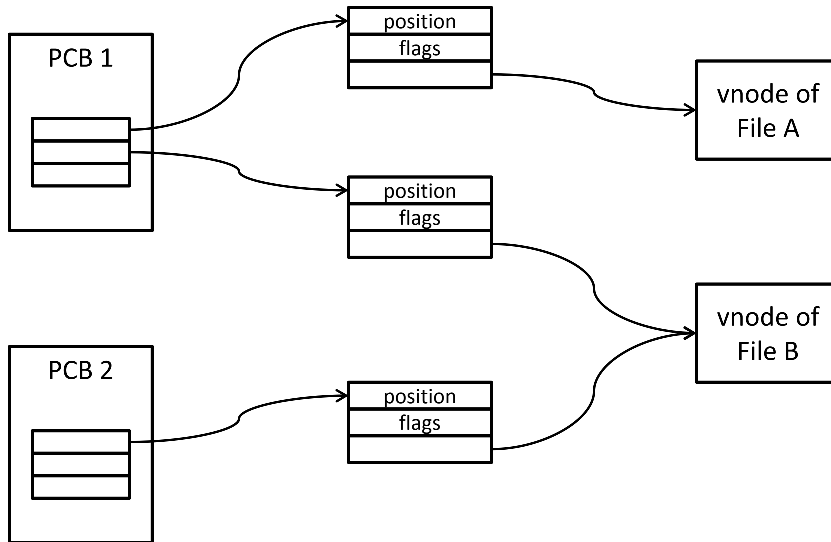
Jeder Prozess hat seinen eigenen Satz an offenen Dateien. Beim Öffnen einer Datei wird ein Eintrag mit einem neuen Dateideskriptor zur lokalen Dateitabelle im PCB hinzugefügt. Zusätzlich wird ein neuer Eintrag in der globalen (systemweiten) Dateitabelle hinzugefügt und der lokale Dateideskriptor mit dem globalen Eintrag verknüpft.

Jeder Eintrag in der globalen Dateitabelle enthält Metadaten zur Datei, wie z.B. die aktuelle Position innerhalb der Datei oder die Zugriffsrechte, mit denen die Datei geöffnet wurde. Weiterhin enthält der Eintrag einen Zeiger auf eine Datenstruktur, die die Datei eindeutig auf dem Dateisystem identifiziert (vNode).

Jeder *open* Systemaufruf erzeugt einen neuen Eintrag in der lokalen und in der globalen Dateitabelle. Eine Datei kann mehrfach geöffnet sein, dann zeigen die Einträge auf die gleiche vNode Struktur.

Tutorien Übungsblatt - File System Basics

PCBs with local open file tables | Global open file table | vnode table



Tutorien Übungsblatt - File System Basics

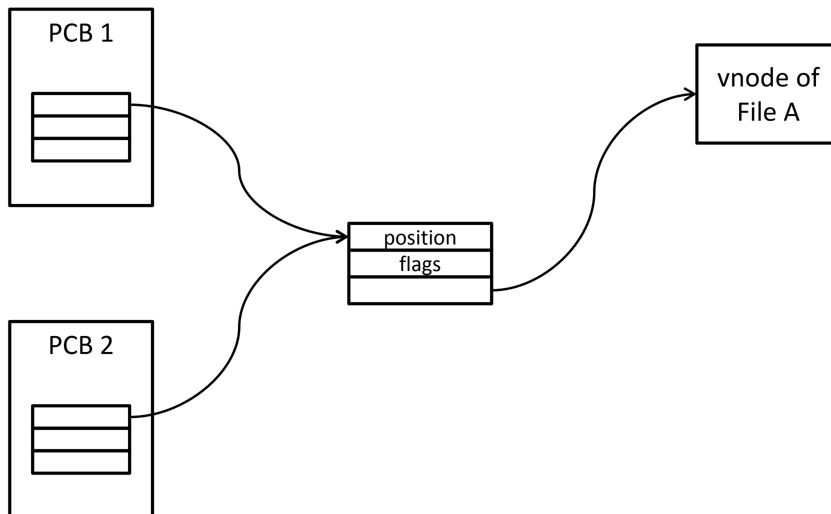
Beispiel: *fork*

Es können auch mehrere Einträge aus der lokalen Dateitabelle auf den gleichen Eintrag in der globalen Dateitabelle zeigen.

Bei einem *fork* Aufruf wird die die lokale Dateitabelle kopiert. Beide Prozessen können die Dateideskriptoren unabhängig voneinander manipulieren, allerdings wirken sich Veränderungen der Metadaten, wie z.B. das Setzen einer neuen Position innerhalb einer Datei, auch im jeweils anderen Prozess aus.

Tutorien Übungsblatt - File System Basics

PCBs with local open file tables | Global open file table | vnode table



Tut. Übungsblatt - Implementing Random Access to Files

Unix und Windows implementieren bei wahlfreiem Zugriff das Setzen einer neuen Position innerhalb einer Datei mit einem Systemaufruf.

Frage 10.2.a

Wie lauten die Namen dieser Systemaufrufe?

Antwort

Unix
lseek

Windows
SetFilePointer/SetFilePointerEx

Frage 10.2.b

Wie unterscheiden sich die Systemaufrufe?

Unix

off_t lseek(int fildes, off_t offset, int whence)

Parameter

- **fildes**: Dateideskriptor
- **offset**: Gewünschter Offset (64 Bit)
- **whence**: Art der Änderung

Rückgabewert: Neue Position (64 Bit)
Art der Änderung

- **SEEK_SET**: Position auf gewünschten Wert setzen (absolut)
- **SEEK_CUR**: Neue Position = Aktuelle Position + Gewünschter Offset (relativ zu aktueller Position)
- **SEEK_END**: Neue Position = Dateilänge + Gewünschter Offset (relativ zu Ende der Datei)

Die Dateiposition kann auch größer als die Länge der Datei sein. Wenn Daten später geschrieben werden wird Nullen in die Lücke geschrieben.

Windows

DWORD SetFilePointer(HANDLE hFile, LONG lDistanceToMove, PLONG lpDistanceToMoveHigh, DWORD dwMoveMethod)

Parameter

- **hFile**: Handle von Datei
- **lDistanceToMove**: Untere 32 Bit des Offsets
- **lpDistanceToMoveHigh**: Obere 32 Bit des Offsets
- **dwMoveMethode**: Ausgangspunkt für die Verschiebung

Rückgabewert: Neue Position (Untere 32 Bit)
Ausgangspunkt für die Verschiebung

- **FILE_BEGIN**: Beginn der Datei
- **FILE_CURRENT**: Aktuelle Position
- **FILE_END**: Ende der Datei

Frage 10.2.c

Wie kann wahlfreier Zugriff ohne diese zusätzlichen Systemaufrufe umgesetzt werden?

Antwort

Es kann ein zusätzlicher Positions Parameter zu den *read* und *write* Systemaufrufen hinzugefügt werden. Dadurch wird Platz im Kern eingespart, da die aktuelle Position nicht mehr gespeichert werden muss, allerdings muss nun die Anwendung die Position speichern. Bei wahlfreiem Zugriff wird ein Systemaufruf eingespart, sehr viele Dateien werden allerdings sequentiell bearbeitet: Dort erhöht sich die Bearbeitungszeit durch den jetzt nötigen seek Befehl vor jeder Lese- oder Schreiboperation.

Tut. Übungsblatt - Implementing Random Access to Files

Frage 10.2.d

Ist der *open* Systemaufruf zwingend nötig?

Antwort

Der *open* Systemaufruf erzeugt ein Kernobjekt, das den Zustand einer offenen Datei kapselt.

Der *open* Systemaufruf kann entfernt werden, allerdings muss dann der Zustand jedem Systemaufruf, der Dateien manipuliert, übergeben werden.

Beispiel: *read* Systemaufruf

Als zusätzliche Argumente müssen der Dateipfad und die Dateiposition übergeben werden. Bei jedem *read* Aufruf muss nun der Pfad interpretiert und das vNode Objekt ermittelt werden. Weiterhin müssen auch jedesmal die Dateirechte ermittelt und die Dateiposition gesetzt werden.

Diese zusätzliche Arbeit erhöht die Laufzeit deutlich und führt zu einer deutlich schlechteren Programmgeschwindigkeit.

Tutorien Übungsblatt - Virtual File System

Frage 10.3.a

Welche Vorteile bietet der Einsatz eines virtuellen Dateisystems (VFS)?

Antwort

Ein VFS implementiert eine sehr abstrakte API für Anwendungsprogramme und für Kernelsysteme. Dabei werden die Unterschiede zwischen den verschiedenen Dateisystemen und den unterschiedlichen Speicherorten verborgen.

Beispiel: Vorteil durch Einsatz von VFS

Jedes Kernelsystem, das auf Dateien zugreifen will, kann mit der gleichen API auf Dateien, die auf einem ext4/ReiserFS/NTFS Dateisystem auf der lokalen Festplatte oder per NFS/SMB auf einem Server gespeichert sind, zugreifen.

Tutorien Übungsblatt - Virtual File System

Frage 10.3.b

Welche Nachteile gibt es bei dem Einsatz eines VFS?

Antwort

Spezielle Fähigkeiten von Dateisystemen können nicht mehr (einfach) genutzt werden.

Um die speziellen Fähigkeiten einsetzen zu können muss das VFS umgangen werden. Bei Unix gibt es dafür z.B. den *ioctl* Systemaufruf. Die Anwendung verliert dabei die Kompatibilität mit anderen Dateisystemen.

Tutorien Übungsblatt - Virtual File System

Frage 10.3.c

Welche Ebene der Kernelsysteme löst symbolische Links bzw. Hardlinks auf?

Antwort

Symbolische Links werden vom VFS aufgelöst:
Das Dateisystem liefert den Inhalt des symbolischen Links zurück (absoluter oder relativer Pfad). Anschließend wird die Namensauflösung mit dem neu errechneten Pfad erneut gestartet.

Hardlinks werden vom Dateisystem zu einem Inode aufgelöst, da sie sich nicht von normalen Dateien unterscheiden.

Tutorien Übungsblatt - Fun with Files in Linux

Frage 10.4.a
Wie kann man einfach eine leere Datei anlegen?

Antwort
touch datei

Frage 10.4.b
Wie kann eine Datei für den aktuellen Benutzer ausführbar gemacht werden?

Antwort
chmod u+x datei

Tutorien Übungsblatt - Fun with Files in Linux

Frage 10.4.c
Was ist der Unterschied zwischen *chmod u+x datei* und *chmod u+X datei*?

Antwort
Der erste Befehl macht die Datei immer für den aktuellen Benutzer ausführbar, der zweite Befehl nur dann, wenn die Datei schon vorher ausführbar war.

Frage 10.4.d
Wie kann mit einem Programm eine Datei mit Loch erzeugt werden?

Antwort
sparsefile.c

Tutorien Übungsblatt - Fun with Files in Linux

Frage 10.4.e
Wie kann herausgefunden werden ob eine Datei ein Loch hat?

Antwort
Wenn die Ausgabe von *du -h --apparent-size* viel größer als die Ausgabe von *du -h* ist handelt es sich vermutlich um eine Datei mit Loch.

Frage 10.4.f
Wie kann mit einem Programm eine Datei kopiert werden?

Antwort
copy.c

Fragen & Kommentare

Fragen & Kommentare?

Ende

xkcd: Supported Features