Betriebssysteme

(Praktikumsdokumentation)

Bachelorstudiengang
Informatik

Sommersemester 2024

Prof. Dr. Martin Pellkofer

Fakultät Informatik

Die Shell

- Die Shell…
  - ist kein Teil des BS, nutzt jedoch stark dessen Eigenschaften
  - die wichtigste Schnittstelle zwischen einem Benutzer, der an seinem Terminal sitzt
  - existieren in vielfältiger Ausprägung: **`sh`**, **`csh`**, **`ksh`**, **`bash`**
    - alle bieten die Funktionalität der ursprünglichen Shell: **`sh`**
- Wenn der Benutzer einen Befehl **`date`** eingibt, erzeugt die Shell einen Kindprozess und lässt das Programm **`date`** als Kindprozess laufen.
  - Während dieser Kindprozess ausgeführt wird, wartet die Shell auf dessen Terminierung.
  - Ist der Kindprozess beendet, dann gibt die Shell erneut das Prompt-Zeichen aus und versucht, die nächste Eingabezeile zu lesen.
- Beispiel für Ausführung in einer Pipe: **`cat file1 file2 file3 | sort >/dev/lp`**
  - Das Programm **`cat`** wird aufgerufen, um die drei Dateien aneinanderzuhängen (*concatenate*).
  - Die Ausgabe wird anschließend durch den Pipe-Operator | an **`sort`** übergeben, um die Zeilen in alphabetischer Reihenfolge zu sortieren.
  - Die sortierte Ausgabe von **`sort`** wird auf die Datei **`/dev/lp`** umgelenkt, die ein typischer Name für eine Zeichendatei zur Ansteuerung eines Druckers ist.

Die Shell (2)

- Wenn der Benutzer ein „**&**“ an ein Kommando anfügt, so wartet die Shell nicht auf dessen Terminierung. Stattdessen wird sofort das Prompt-Zeichen ausgegeben.
- Diese GUI ist eigentlich nur ein Programm, das über dem Betriebssystem läuft, genau wie eine Shell.
  - In Linux-Systemen ist dies sehr offensichtlich, weil der Benutzer zwischen min. 2 GUIs wählen kann: Gnome oder KDE.

Systemaufrufe für Dateioperationen

- **`create`**
  - Datei ohne Daten erzeugen
  - Entstehung der Datei ankündigen
  - einige Attribute festlegen
- **`delete`**
  - Datei löschen und Speicherplatz auf Datenträger freigeben
  - ist immer vorhanden
- **`open`**
  - Datei öffnen
  - die Attribute und die Liste der Plattenadressen werden in Arbeitsspeicher geladen → Tabellenspeicher
- **`close`**
  - Datei schießen und die internen Tabellenspeicher freigeben
  - Schließen erzwingt das Schreiben des letzten Blocks der Datei
- **`read`**
  - Daten werden (gewöhnlich von der aktuellen Position) gelesen
  - Aufrufer muss die Anzahl der Bytes angeben und Puffer für Daten zur Verfügung stellen.

Systemaufrufe für Dateioperationen (2)

- **`write`**
  - Daten werden (gewöhnlich an die aktuelle Position) geschrieben.
  - wenn aktuelle Position das Ende der Datei ist → Dateigröße erhöht sich
  - wenn aktuelle Position in der Mitte der Datei ist → vorhandene Daten werden überschrieben
- **`append`**
  - ist eingeschränkte Form von **`write`**
  - kann nur benutzt werden, um Daten an des Ende der Datei zu schreiben
- **`seek`**
  - positioniert den Datenzeiger an bestimmte Stelle in der Datei
  - wird für wahlfreien Zugriff benutzt
- **`get attributes`**
  - Dateiattribute lesen
  - Beispielanwendung: minimaler Übersetzungsvorgang mit **`make`**
- **`set attributes`**
  - Dateiattribute verändern
  - Attribute besitzen zweistellige Wertigkeit → werden jeweils durch genau 1 Bit (Flag) repräsentiert
  - Beispiel: Schutzstatus
- **`rename`**
  - Dateinamen verändern
  - ist nicht zwingend nötig

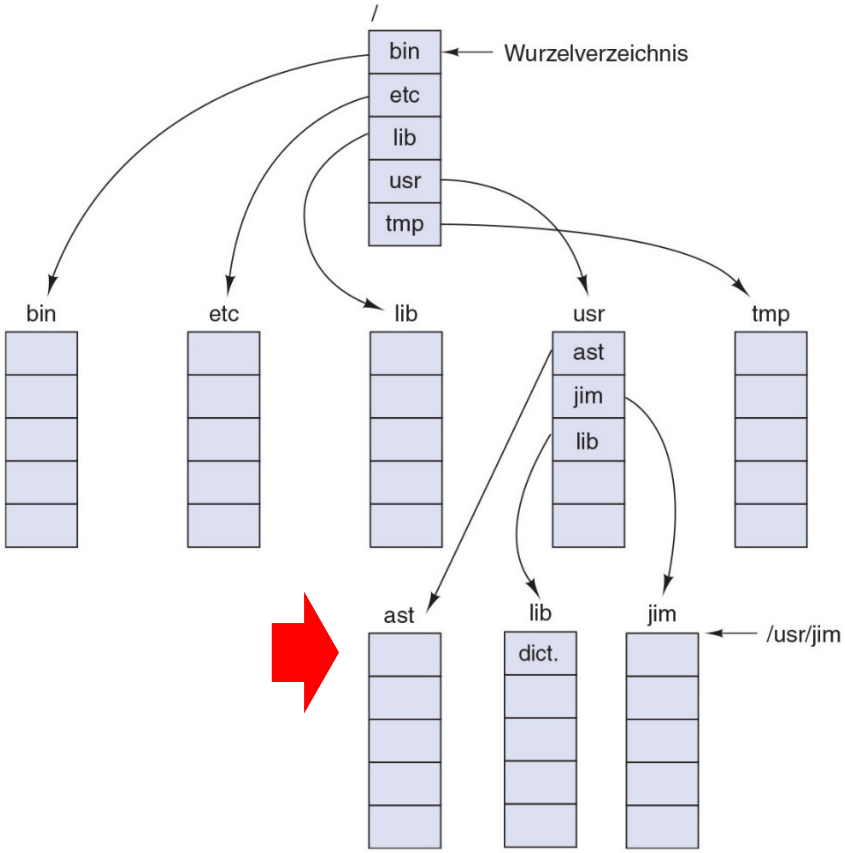
Systemaufrufe auf Verzeichnisse

- **`create`**
  - Verzeichnis anlegen
  - dieses ist bis auf `.` und `..` leer
- **`delete`**
  - Verzeichnis löschen
  - nur ein leeres Verzeichnis kann gelöscht werden
- **`opendir`**
  - öffnen des Verzeichnisses und internen Tabellenspeicher laden
  - um z. B. die Namen aller Dateien in einem Verzeichnis auflisten zu können
- **`closedir`**
  - schließen eines Verzeichnisses
  - gibt interne Tabellenspeicher frei
- **`readdir`**
  - gibt den nächsten Eintrag eines geöffneten Verzeichnisses zurück
  - gibt immer einen Eintrag in einem Standardformat zurück.
- **`rename`**
  - Verzeichnisses umbenennen
- **`link`**
  - Datei verlinken (*hard link*)
  - damit kann dieselbe Datei in mehreren Verzeichnissen vorkommen
- **`unlink`**
  - Verzeichniseintrag entfernen
  - wenn Datei dadurch in keinem Verzeichnis mehr vorhanden ist, wird sie aus dem Dateisystem entfernt

readdir

Die Einträge "." und ".."

- Die meisten Systeme mit hierarchischer Verzeichnisstruktur haben 2 spezielle Einträge in jedem Verzeichnis um relative Pfadangaben zu ermöglichen:
  - "." (sprich "*Punkt*")
    - bezieht sich auf das aktuelle Verzeichnis
  - ".." (sprich "*Punktpunkt*")
    - bezieht sich auf das übergeordnete Verzeichnis
    - Ausnahme: Im Wurzelverzeichnis verweist ".." auf sich selbst.
- Falls Arbeitsverzeichnis `/usr/ast` ist, bewirken folgende Befehle immer das Gleiche:
  - `cp ../lib/dictionary .`
  - `cp /usr/lib/dictionary .`
  - `cp /usr/lib/dictionary dictionary`
  - `cp /usr/lib/dictionary /usr/ast/dictionary`



Quelle: [Tanenbaum und Bos, 2016]

POSIX-Systembefehl für das Verlinken von Dateien

- Der Systemaufruf **`link`** erlaubt es einer Datei, unter verschiedenen Namen in unterschiedlichen Verzeichnissen vorzukommen.
  - typische Anwendung: gemeinsame Nutzung der Datei durch mehrere Mitgliedern einer Gruppe
- Beispiel für Anwendung von **`link()`**:
  - Es gibt 2 Benutzer **`ast`** und **`jim`** und jeder besitzt sein eigenes Verzeichnis mit Dateien.
  - Benutzer **`ast`** führt nun ein Programm aus, das folgenden Aufruf enthält:

```
link("/usr/jim/memo", "/usr/ast/note");
```

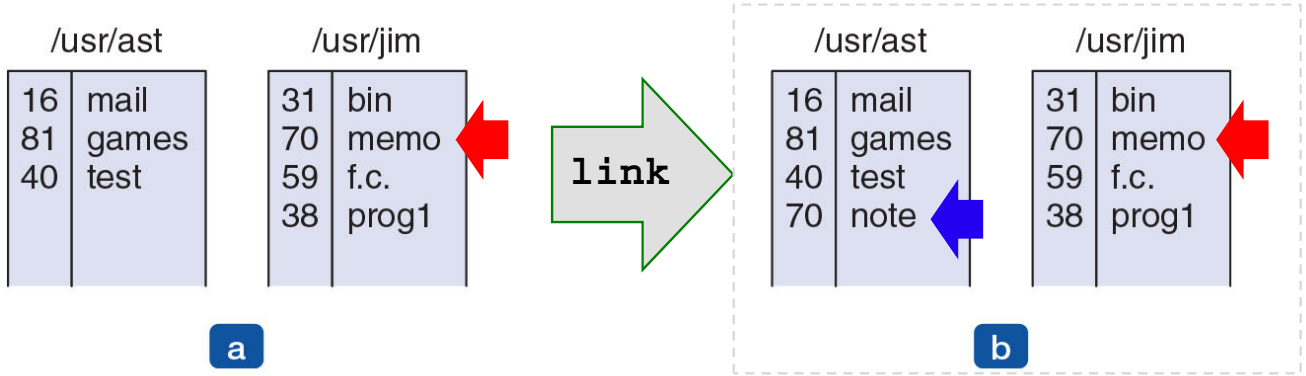
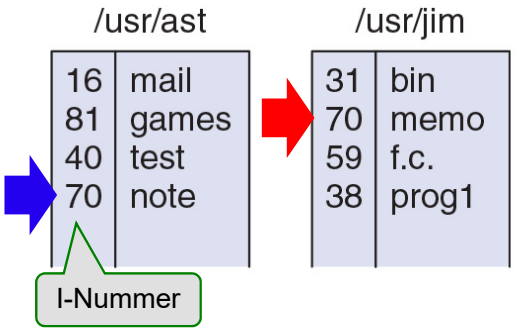


Abbildung 1.21: (a) Zwei Verzeichnisse, bevor */usr/jim/memo* in das Verzeichnis *ast* verlinkt wurde; (b) dieselben Verzeichnisse nach dem Aufruf von `link`.

Quelle: [Tanenbaum und Bos, 2016]

Einwurf: I-Nodes und I-Nummer bei UNIX



- Jede Datei unter UNIX besitzt zur Identifizierung eine eindeutige Nummer, die sogenannte I-Nummer.
  - Diese I-Nummer ist ein Index in einer Tabelle mit I-Nodes.
  - Jeder I-Node gibt für jede Datei an, wem die Datei gehört, wo die Blöcke auf der Platte liegen und so weiter.

- Ein Verzeichnis ist somit (prinzipiell) lediglich eine Datei mit einer Menge von Paaren aus I-Nummer und ASCII-Name.
  - Der Systemaufruf **`link`** erzeugt nun einfach einen völlig neuen Verzeichniseintrag mit einem (möglichst neuen) Namen, der die I-Nummer von einer existierenden Datei verwendet.
  - Wenn später einer der beiden mit dem **`unlink`**-Systemaufruf gelöscht wird, bleibt der andere Eintrag bestehen.
  - Wenn alle Verzeichniseinträge gelöscht werden, erkennt UNIX, dass keine Einträge für diese Datei mehr bestehen und löscht die Datei.
    - Dazu zählt ein Eintrag in der I-Node-Tabelle die Anzahl der aktuellen Zeiger auf diese Datei mit.

Quelle: [Tanenbaum und Bos, 2016]

Symbolischer Link

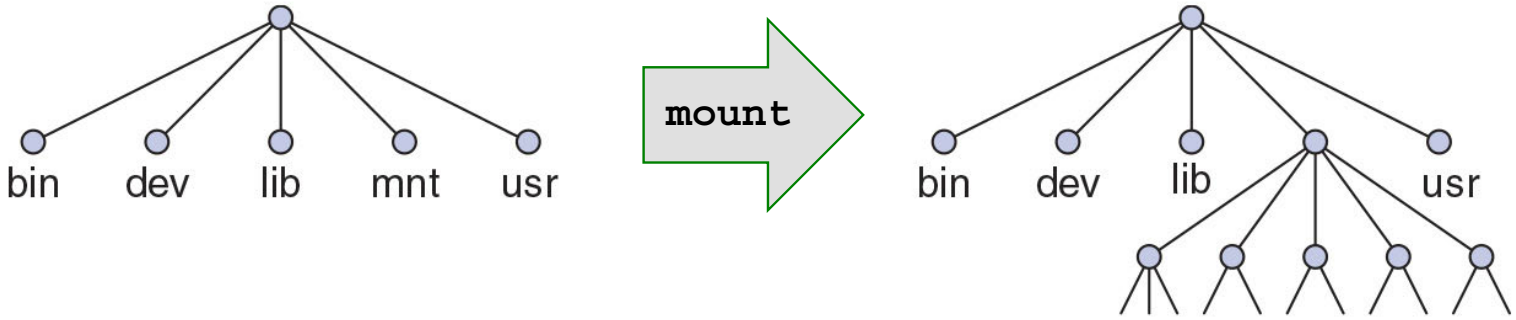
- Eine Variante des Konzepts zur Verlinkung von Dateien ist der symbolische Link:
  - Vorgehen:
    - Es wird eine kleine Datei mit einem Namen erzeugt, welche wiederum den Namen einer anderen Datei (mit Pfad) enthält.
    - Wenn auf die erste Datei zugegriffen wird, folgt das Dateisystem dem darin enthaltenen Pfad und findet an dessen Ende den Namen der zweiten Datei.
    - Dann wird der Zugriff nochmals gestartet, diesmal unter Verwendung des neuen Namens.
  - Vorteil: können Plattengrenzen überschreiten und sogar Dateien auf entfernten Computern ansprechen
  - Nachteil: Implementierung ist weniger effizient als bei harten Links

POSIX-Systembefehle für das *Mounten* von Dateisystemen

- Der **`mount`**-Systemaufruf fasst 2 Dateisysteme zu einem zusammen.
- Beispiel: Dateisystem des USB-Laufwerks Nr. 0 in ein Unterverzeichnis des Wurzelverzeichnisses einhängen:

```
mount("/dev/sdb0", "/mnt", 0);
```

1. Param.: Name der Blockdatei (= Spezialdatei für Plattengerät) für das USB-Laufwerk mit Nr. 0
2. Param.: Ordner in welchen Verzeichnisbaum eingehängt wird.
3. Param.: Zugriffsart Lesen und/oder Schreiben



  - Nach dem Aufruf von **`mount`** kann jede Datei von Laufwerk **`sbd0`** durch Angabe des Pfads vom Wurzelverzeichnis oder vom Arbeitsverzeichnis aus angesprochen werden.
- Wenn ein Dateisystem nicht mehr gebraucht wird, kann es einfach mit dem **`umount`**-Systemaufruf ausgehängt werden.

Quelle: [Tanenbaum und Bos, 2016]

Datenschutz und Datensicherzeit im Dateisystem

- Computer verwalten eine große Menge von Informationen, die der Benutzer oft schützen und vertraulich behandeln möchte.
- Es ist die Aufgabe des BS, die Systemsicherheit zu wahren, sodass zum Beispiel Dateien nur von autorisierten Benutzern gelesen werden können.
- Dateien unter UNIX werden mit einem 9-Bit-Code geschützt (**`rwx`**-Bits):
  - Dieser Code besteht aus drei 3-Bit-Feldern:
    { *Eigentümer*, *Inhabergruppe*, *alle anderen Benutzer* }
  - Jedes Feld wiederum besitzt ein Bit für den Lesezugriff, eines für den Schreibzugriff und eines für die Berechtigung, die Datei auszuführen → **`rwx`**-Bits (von *read*, *write* und *execute*).
  - Beispiel für 9-Bit-Code: **`rwxr-x--x`**
    - Ein „-“ steht immer für ein fehlendes Recht.
  - Bei einem Verzeichnis beeinflusst ...
    - das **`r`**-Bit die Erlaubnis zum Auflisten der Dateien im Verzeichnis
    - das **`x`**-Bit die Erlaubnis für einen Zugriff auf die Dateien, ggf. auch ohne das Recht zum Auflisten
    - das **`w`**-Bit die Erlaubnis, eine Datei aus dem Verzeichnis löschen zu dürfen

Funktionen zum Ändern:
chmod, chown, chgrp