13. Temporäre Ramdisk in der Mini-Shell

Die Mini-Shell soll nun auch eine eingebaute Ramdisk unterstützen, welche die temporäre Ablage von Dateien im Hauptspeicher (genauer: im Speicher des Shell-Prozesses) erlaubt. Dazu implementieren Sie acht neue interne Shell-Kommandos, welche die Shell erkennt und selbst ausführt (also kein externes Programm startet):

a) `rdinit`: Dieses Kommando initialisiert die Ramdisk. Wenn es mehr als einmal aufgerufen wird, gibt es ab dem zweiten Aufruf eine Fehlermeldung („Ramdisk bereits initialisiert“) aus. Erst nach dem Aufruf von `rdinit` werden die folgenden Kommandos akzeptiert.

b) `rdget Pfad Dateiname`: Kopiert die über `Pfad` angesprochene Datei in die Ramdisk (und verwendet in der Ramdisk den Namen `Dateiname`). Dateiattribute (User/Group, Zugriffsrechte) sollen dabei mit gespeichert werden. Bereits in der Ramdisk vorhandene Dateien werden bei Namensgleichheit auf Nachfrage („Wirklich überschreiben? j/n“) überschrieben.

c) `rdls`: Zeigt ein Inhaltsverzeichnis der Ramdisk (im Stil von `ls -l`) an, am Ende der Liste erscheint außerdem zusammenfassend der im Hauptspeicher belegte Platz. Wenn Sie geeignete Datenstrukturen erzeugen, können Sie hier mit kleinen Änderungen Ihre `ls()`-Funktion verwenden.

d) `rdput Dateiname Pfad`: Kopiert die unter `Dateiname` in der Ramdisk abgelegte Datei auf die Platte (dort nach `Pfad`), auch hier werden die Dateiattribute wieder übernommen. Wenn Sie nacheinander `getdisk Pfad tmp` und `putdisk tmp Pfad2` ausführen, soll dies also dem Kopieren einer Datei von `Pfad` nach `Pfad2` entsprechen.

e) `rdrm Dateiname`: löscht die angegebene Datei aus der Ramdisk.

f) `rdln Datei Link`: erzeugt intern in der Ramdisk einen Hardlink von `Datei` auf `Link`.

g) `rdmv AlterName NeuerName`: benennt in der Ramdisk eine Datei um, falls `NeuerName` noch nicht existiert.

h) `rdchmod Rechte Dateiname`: Setzt die Rechte für die Datei. Die Rechte werden in Oktalnotation angegeben (z. B. `rdchmod 660 Datei`).

Alle benötigten Datenstrukturen erzeugen Sie bei Bedarf mit `malloc` oder `calloc`; nach dem Löschen einer Datei aus der Ramdisk geben Sie den nicht mehr benötigten Speicherplatz mit `free` frei.

Für die Organisation der Ramdisk verwenden Sie die folgenden Datenstrukturen:

- Bei der Initialisierung erzeugen Sie eine Inode-Liste, die Platz für 50 Inodes bereithält.
- Jede Datei in der Ramdisk wird über einen dieser Inodes angesprochen. Im Inode speichern Sie einen Zeiger auf den Speicherbereich, der die Datei enthält, die Größe der Datei und ihre Attribute sowie den Link Count (Anzahl der vergebenen Namen).
- Das Inhaltsverzeichnis der Ramdisk ist eine Liste von Paaren (Dateiname, Inode-Nummer). Beachten Sie beim Löschen einer Datei (`rdrm`), dass der Dateiinhalt erst dann entfernt und der Inode freigegeben werden darf, wenn der Link Count den Wert 0 erreicht hat.

Testen Sie Ihr Programm, indem Sie einige kleinere Dateien in die Ramdisk kopieren, dort manipulieren (umbenennen, Hard Links erzeugen, Rechte ändern etc.) und dann wieder ins Dateisystem zurück kopieren. Sie können zwei Dateien mit dem Tool `diff` vergleichen. Dateikopien über die Ramdisk sollten inhaltsgleich sein und so viele Attribute wie möglich erhalten, sofern Sie diese nicht mit `rdchmod` geändert haben. (Nur der Administrator `root` kann beim Kopieren von Dateien alle Attribute erhalten.)

Alternativ können Sie `rdget`, `rdput` und `rdmv` zu einem Kommando `rdmv` zusammenfassen, wobei dann Dateinamen in der Ramdisk über ein Präfix `rd:` kenntlich gemacht werden.

Lösungshinweise (Spoiler)

Ein **Inode** speichert die Metadaten (Zugriffsrechte, Besitzer/Gruppe, drei Zeitstempel, Dateigröße, Verweise auf Datenblöcke) einer Datei – bei der Ramdisk brauchen Sie keine Datenblock-Verweise, die übrigen Daten aber schon.

Wenn Sie mit

```
struct stat status;
stat(filename, &status);
```

die Statusinformationen einer Datei (auf Platte) auslesen, haben Sie bereits alle benötigten Informationen in der Variable `status`. Mit `man 2 stat` sehen Sie den Aufbau des Datentyps `struct stat`; u. a. steht in `status.st_size` die Dateigröße – das ist die Menge an Speicher, die Sie mit `malloc()` reservieren müssen, damit Sie die Datei vollständig in den Speicher (in Ihre Ramdisk) kopieren können. Legen Sie einfach für jede neue Datei einen neuen Speicherbereich an, beim Löschen einer Datei geben Sie diesen wieder frei.

Wenn Sie ein Beispiel für eine „echte“ Inode-Struktur (für ein Festplatten-Dateisystem) sehen wollen, googeln Sie „minix file:inode.h“ und schauen Sie sich in der gefundenen C-Headerdatei `inode.h` die Definition des Datentyps `struct inode` an.

Definieren Sie einen Datentyp `struct inode`, der alle Inode-Felder enthält. Dann können Sie ein Array `inode_table` erzeugen, in das 50 Inodes passen.

Die von `malloc()` zurückgelieferte Speicheradresse sollten Sie im Inode speichern (Sie brauchen dafür ein zusätzliches Feld, das eine Adresse aufnehmen kann). Dann können Sie beim Löschen der Datei `free()` mit dieser Adresse aufrufen, um den Speicher wieder freizugeben.

Achten Sie bei Aufgabe e) darauf, dass Sie eine Datei beim Aufruf von `rdrm` nur dann löschen, wenn der Link Count des Inodes durch auf 0 sinkt – anderenfalls gibt es nämlich noch weitere (mit `rdln`) erzeugte Namen für diese Datei. Das entspricht dem normalen Linux-/Unix-Verhalten, weshalb intern die Löschfunktion auch `unlink()` und nicht `remove()` oder ähnlich heißt.