Hochschule RheinMain
DCSM - Informatik
Prof. Dr. Reinhold Kröger

Verteilte Systeme
WS 2015/16

Übungsblatt 1
Praktische Übungen

Die folgenden Übungsaufgaben erfordern Grundkenntnisse der C-Programmierung und dienen der Wiederholung. Ziel dieser Übung ist die Implementierung eines einfachen, virtuellen Dateisystems, das als Grundlage für weitere Übungen vorgesehen ist.

Hinweis:
In diesem Praktikum werden Abgaben in einem zentralen SVN-Repository verwaltet. Die Praktikumsleiter legen dazu für jedes Übungsblatt eine Vorlage an, die Sie am Anfang der Übung auschecken müssen. Machen Sie sich mit der der Handhabung von SVN vertraut. Weiterführende Hilfe finden Sie dazu im Wiki des Studiengangs und im Internet, z.B.:
`https://www-intern.cs.hs-rm.de/publicwiki/index.php/Hauptseite`

In den Vorlagen finden Sie eine einheitliche Projektstruktur vor, die Sie nicht verändern dürfen. Das Projekt beinhaltet i.d.R. Templates für alle Quelldateien und die anzulegende Dokumentation. Für C-basierte Aufgaben wird außerdem ein Makefile vorgegeben, für Java-basierte Aufgaben wird ein Eclipse-Projekt vorgegeben.

`https://svn.cs.hs-rm.de/svn/vs15_mmust001/1/`	Vorlage Aufgabenblatt 1
`https://svn.cs.hs-rm.de/svn/vs15_mmust001/2/`	Vorlage Aufgabenblatt 2
`https://svn.cs.hs-rm.de/svn/vs15_mmust001/3/`	Vorlage Aufgabenblatt 3

Initialer Checkout der Aufgaben aus dem Repository:

```
$ svn checkout https://svn.cs.hs-rm.de/svn/vs15_mmust001/
```

Aktualisierung des Repositories:

```
$ svn update
```

Anzeigen der lokalen Änderungen:

```
$ svn diff
```

`Speichern der lokalen Änderungen im` Repository (wichtig!):

```
$ svn commit
```

Aufgabe 1 (Projekt “VFileSystem”):
Implementieren Sie die Funktionen der vorliegenden FileSystem-API (vfilesystem_api.h) in der vorbereiteten Datei (vfilesystem_api.c).

Das Dateisystem bietet die Möglichkeit, Operationen auf Dateien und Verzeichnissen auszuführen. Beachten Sie, dass sämtliche Dateien und Verzeichnisse im Arbeitsspeicher zu halten sind und keine Repräsentierung im Hintergrundspeicher besitzen. Für die Referenzierung werden Folder- und FileIDs verwendet.

Das Dateisystem bietet folgende Funktionalität:

`FileID fs_new_file(char *name, FolderID parent)`:

Erstellt eine neue Datei mit dem Namen `name` in dem Verzeichnis mit der `FolderID parent` und gibt die `FileID` der neu erzeugten Datei zurück.
return:

- `FileID` bei Erfolg
- `INVALID_HANDLE`, falls `parent` nicht existiert
- negativer Wert sonst

Gehen Sie sicher, dass es nicht möglich ist, Dateien mit identischem Namen im selben Verzeichnis zu erzeugen.

`FolderID fs_new_folder(char *name, FolderID parent)`:

Erstellt ein neues Verzeichnis mit dem Namen `name` in dem Verzeichnis mit der `FolderID parent` und gibt die `FolderID` des neu erzeugten Verzeichnisses zurück.
return:

- `FolderID` bei Erfolg
- `INVALID_HANDLE`, falls `parent` nicht existiert
- negativer Wert sonst

Gehen Sie sicher, dass es nicht möglich ist, ein Verzeichnis mit identischem Namen im selben Verzeichnis zu erzeugen.

`int32_t fs_delete_file(FileID file):`

Entfernt die Datei mit der `FileID file` aus dem Dateisystem.
return: 0 bei Erfolg, -1 im Fehlerfall.

`int32_t fs_delete_folder(FolderID folder):`

Entfernt das Verzeichnis mit der `FolderID folder` aus dem Dateisystem.
return: 0 bei Erfolg, -1 im Fehlerfall.

```
FolderID fs_get_file_parent(FileID file):
FolderID fs_get_folder_parent(FolderID folder):
```

Liefert das Verzeichnis, in dem eine Datei bzw. ein Verzeichnis liegt.

`int32_t fs_get_file_size(FileID file):`

Liefert die Größe einer Datei.

```
int32_t fs_get_file_name_length(FileID file):
int32_t fs_get_folder_name_length(FolderID folder):
```

Liefert die Länge eines Dateinamens bzw. Verzeichnisnamens.

```
int32_t fs_get_file_name(FileID file, char* file_name, uint8_t max_length):
int32_t fs_get_folder_name(FolderID folder, char* folder_name,
    uint8_t max_length):
```

Schreibt den Namen einer Datei bzw. eines Verzeichnisses in einen Puffer (der vorher ausreichend groß allokiert werden muss).

```
int32_t fs_get_folder_file_count(FolderID folder):
int32_t fs_get_folder_folder_count(FolderID folder):
```

Liefert die Anzahl von Dateien bzw. Verzeichnissen in einem Verzeichnis.

```
int32_t fs_get_folder_files(FolderID folder, FileID* files,
      uint32_t max_num_files):
int32_t fs_get_folder_folders(FolderID folder, FolderID* folders,
      uint32_t max_num_folders):
```
Schreibt die IDs von Dateien bzw. Verzeichnissen innerhalb eines Verzeichnisses in einen Puffer (der vorher ausreichend groß allokiert werden muss).

```
int32_t fs_write_file(FileID file, u_int32_t offset, u_int32_t length,
u_int8_t *data):
```
Schreibt `length` Bytes aus dem Buffer `data` in die Datei `file` beginnend bei Offset `offset.`

return: 0 bei Erfolg, -1 im Fehlerfall.

```
int32_t fs_read_file(FileID file, u_int32_t offset, u_int32_t length,
u_int8_t *data):
```
Liest `length` Bytes aus der Datei `file` in den Buffer `data` beginnend bei Offset `offset.`

return: 0 bei Erfolg, -1 im Fehlerfall.

Schreiben Sie Aufrufe zum Testen der Dateisystem-Funktionen in die `main`-Funktion.

Analysieren Sie ihr Programm mithilfe von valgrind [1] und beseitigen Sie eventuelle Fehler.

[1] http://valgrind.org/docs/manual/manual.html