HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Institut für Informatik
Prof. Dr. Martin Mauve
Rechnernetze und Kommunikationssysteme

Grundlagen der Technischen Informatik SoSe 2013
C-Programmieraufgabe

1 Organisatorisches

Schauen Sie regelmäßig auf die Website zum C-Projekt[1]. Dort werden eventuell kleine Änderungen der Aufgabenstellung oder zusätzliche Hilfestellungen veröffentlicht.

Implementieren Sie in **Einzelarbeit** ein Programm in der Programmiersprache C, das die folgende Problemstellung löst. Bitte wenden Sie sich bei Fragen zur Aufgabenstellung wie gewohnt an den *Tutor Ihrer Gruppe* oder an die Mitarbeiter des Lehrstuhls: *Norbert Goebel*[2] oder *Matthias Radig*[3].

Plagiate führen zum Auschluss von der Veranstaltung.

Das Lösen der Programmieraufgabe ist entscheidend für die Zulassung zur Klausur. Jede abgegebene Lösung wird in einem kurzen Kolloquium besprochen.

Es gibt drei Abgabetermine, die darüber entscheiden, wann sie frühestens an einem Kolloqiumstermin teilnehmen können (siehe Tabelle). An welcher Klausur Sie teilnehmen können hängt wiederum von Ihrem Kolloquiumstermin ab. Sie werden erst nach bestehen des Kolloquiums zur Klausur zugelassen, daher können Sie an einer Klausur nur teilnehmen, wenn Sie vorher Ihr Kolloquium bestanden haben.

spätester Abgabetermin	**frühester Zeitraum für das Kolloquium**	**möglicher Klausurtermin**
07.07.2013	15.07.2013 bis 19.07.2013	Hauptklausur, Nachklausur
08.09.2013	30.9.2013 bis 04.10.2013	Hauptklausur, Nachklausur
14.10.2013	04.11.2013 bis 08.11.2013	Nachklausur

Abgaben nach dem **14.10.2013** können nicht mehr berücksichtigt werden (mit Ausnahme von krankheitsbedingten Gründen). Wenn Sie an der Hauptklausur teilnehmen wollen muss Ihre Abgabe bis zum **08.09.2013** erfolgen und Ihr Kolloquium muss vor der Klausur erfolgreich abgeschlossen sein. Bitte beachten Sie für die Anmeldung zur Klausur umbedingt die Anmeldefristen des Prüfungsamtes. Sie können sich dort auch anmelden wenn Ihr Kolloquiumstermin noch nicht stattgefunden hat und sie daher die Zulassungsvoraussetzungen noch nicht komplett erfüllen.

Wir empfehlen Ihnen eine erste Abgabe mindestens zwei Wochen vor dem endgültigen Termin, so dass wir Ihnen frühzeitig Rückmeldung geben können und Sie bis zum endgültigen Abgabetermin Ihr Programm gegebenenfalls noch einmal nachbessern können. Dies gilt insbesondere für den dritten Abgabetermin (14.10.2013). Geben Sie **spätestens bis zum 01.10.2013 eine erste Fassung** Ihres

[1] http://www.cn.uni-duesseldorf.de/teaching/sose13/info2/folien/c-projekt
[2] Sprechstunde Montags 13:30 - 14:30 in Raum 25.12.02.41, `goebel@cs.uni-duesseldorf.de`
[3] Sprechstunde Freitags 13:30 - 14:30 in Raum 25.12.02.45?, `radig@cs.uni-duesseldorf.de`

Programms ab! Sie dürfen gerne auch jederzeit früher eine Version abgeben und sich Feedback einholen.

Rechnen Sie damit, dass ein ungeübter C-Programmierer zwei bis drei Wochen Vollzeit zur Lösung der Aufgabenstellung benötigt. Fangen Sie rechtzeitig an!

Für das erfolgreiche Bestehen der C-Programmieraufgabe müssen Sie zwei Kriterien erfüllen:

(a) Ihr Programm muss die gestellten Anforderungen erfüllen (insbesondere Abschnitt 4) und korrekt funktionieren. Wir überprüfen es anhand einiger Testdatensätze und automatisierter Tests z. B. auf Speicherlecks.

(b) Sie müssen uns im Kolloquium Ihr Programm und Ihren Algorithmus erläutern.

Das System zur automatischen Überprüfung Ihrer Abgaben steht voraussichtlich ab dem 18.06.2013 zur Verfügung. Sobald das System zur Verfügung steht werden wir die genauen Abgabemodalitäten auf der Website zum C-Projekt[4] veröffentlichen.

2 Aufgabenstellung

Eine häufig wiederkehrende Aufgabe in der Informatik ist das Speichern von Informationen. Dies geschieht im allgemeinen in Dateisystemen, die neben den eigentlichen Dateien auch Verwaltungsinformationen speichern. In dem diesjährigen C-Projekt sollen sie daher Ihr eigenes virtuelles Dateisystem (VFS) implementieren.

Um die Aufgabe zu vereinfachen und Datenverlust vorzubeugen ist der Funktionsumfang des zu implementierenden Dateisystems beschränkt:

- Beim Anlegen einer neuen Instanz Ihres VFS einer bestimmten Größe müssen zwei Dateien auf dem normalen Dateisystem (*Host-Dateisystem*) angelegt werden.
  - Eine dieser Dateien dient alleine der Speicherung der Inhalte der zu speichernden Dateien (im weiteren *vfs.store* genannt). Die Größe dieser Datei ist durch Parameter beim Erstellen des Dateisystems fest vorgegeben und darf nicht varieren.
  - Die andere Datei wird zur Speicherung der Verwaltungsdaten verwendet (im weiteren *vfs.structure* gennant).
- Werden dem VFS Dateien hinzugefügt, werden die Verwaltungsdaten (wie Dateiname, Größe, usw.) in *vfs.structure* gespeichert, die eigentlichen Daten in *vfs.store*.
- Dadurch ist es auf einfache Weise möglich die Verwaltungsdatei wachsen oder schrumpfen zu lassen während *vfs.store* permanent die gleiche Größe behält.
- Wie Sie die Speicherverwaltung und Speicherung der Verwaltungsdaten implementieren ist Ihnen überlassen.
- Der Rückgabewert (*Rückgabewert* bzw. *exit code*) Ihres Programms wird dazu genutzt Fehlerzustände mitzuteilen (Vorgabe in Abschnitt 3 beachten).
- Im VFS gibt es keine Verzeichnisse.
- Achtung: Ein direktes Arbeiten auf raw devices ist nicht nötig und auch nicht erlaubt.

[4] http://www.cn.uni-duesseldorf.de/teaching/sose13/info2/folien/c-projekt

3 Programmoptionen und Parameter

Ihr Programm mit dem Namen **vfs** muss folgende Parameter akzeptieren:

```
> ./vfs ARCHIVE OPERATION [PARAMETERS]
  ARCHIVE             Pfad und Dateiname (ohne Endung) zu den Dateien, die das
                      VFS enthalten/verwalten
  OPERATION:
             create   ein neues VFS anlegen
             add      eine Datei zum VFS hinzufügen
             del      ein Datei aus dem VFS löschen
             defrag   Defragmentierung des VFS
             free     den freien Speicher (in Byte) des VFS ausgeben
             get      eine Datei aus dem VFS lesen
             list     eine Liste der gespeicherten Dateien ausgeben
             used     den belegten Speicher (in Byte) des VFS ausgeben

  PARAMETERS          Parameter, die von der ausgewählten Operation benötigt werden
```

Die folgenden Abschnitte behandeln die einzelnen Operationen im Detail. Bitte halten sie sich unbedingt exakt an die hier gemachten Vorgaben unter Beibehaltung der Reihenfolge der einzelnen Parameter, da Ihre Abgabe maschinell geprüft wird.
Für jeden Befehl sind eine Reihe Fehlercodes definiert (siehe unten). Für jeden weiteren, nicht hier aufgelisteten Fehler verwenden Sie bitte den Rückgabewert `66`**.**

3.1 ./vfs ARCHIVE create BLOCKSIZE BLOCKCOUNT

create soll ein VFS erstellen. Die zugehörigen Dateien werden dabei (falls möglich) mit den Dateinamen *ARCHIVE.store* bzw. *ARCHIVE.structure* im Host-Dateisystem angelegt. Dabei ist *ARCHIVE.store* exakt $BLOCKSIZE \cdot BLOCKCOUNT$ Bytes groß.
Beim Anlegen wird das VFS initialisiert — die von Ihnen gewählten Verwaltungsstrukturen werden erstellt und in *ARCHIVE.structure* gespeichert. Der erste Block im VFS wird als Block 0 bezeichnet, der nächste als Block 1 usw.

Diese Operation benötigt folgende Parameter:

Parameter	**Beschreibung**
BLOCKSIZE	Die Größe eines Speicherblocks in Ihrem Dateisystem in Bytes.
BLOCKCOUNT	Die Anzahl der Speicherblöcke der Größe BLOCKSIZE, die Ihr Dateisystem zur Verfügung stellen soll.

Der Aufruf kann folgende Werte zurückliefern:

Rückgabewert	**Bedeutung**
0	Das Kommando wurde fehlerfrei ausgeführt.
1	Das Anlegen des VFS war nicht möglich. (Aus Gründen, die das Hostsystem und nicht Ihr Programm verursacht hat!)
3	Ein VFS mit dem gegebenen Namen existiert bereits.

Beispiel für den Aufruf:

```
./vfs /home/student/xyz create 4096 40
```

Legt zwei Dateien */home/student/xyz.store* und */home/student/xyz.structure* an.
Dabei hat */home/student/xyz.store* die Größe $4096 \cdot 40 = 164840$ Bytes.
/home/student/xyz.structure speichert die Verwaltungsdaten für das VFS mit Blockgröße=4096 und Blockanzahl=40.

Rechnen Sie damit, dass wir für Blocksize und Blockcount auch mit Werten jenseits von 65536 testen.

3.2 ./vfs ARCHIVE add SOURCE TARGET

add fügt dem VFS eine neue Datei hinzu.

Diese Operation benötigt folgende Parameter:

Parameter	**Beschreibung**
SOURCE	Die Datei (Dateiname incl. Pfad), die in das VFS kopiert werden soll.
TARGET	Der Name, den die Datei im VFS haben soll.

Der Aufruf kann folgende Werte zurückliefern:

Rückgabewert	**Bedeutung**
0	Das Kommando wurde fehlerfrei ausgeführt.
2	Das VFS konnte nicht geöffnet werden.
11	Eine Datei mit diesem Dateinamen existiert bereits (existierende Dateien dürfen nicht überschrieben werden).
12	Nicht genügend freier Speicherplatz im VFS.
13	Die Quelldatei existiert nicht.

Beispiele für den Aufruf:

```
./vfs /home/student/archiv add /home/student/source Zieldateiname
```

Versucht die Datei */home/student/source* im VFS, das durch */home/student/archiv* repräsentiert wird, unter dem Namen *Zieldateiname* zu speichern.

```
./vfs ./archiv add ../another_source Zieldateiname
```

Versucht die Datei *../another_source* im VFS, das durch */home/student/archiv* repräsentiert wird, unter dem Namen *Zieldateiname* zu speichern.

3.3 ./vfs ARCHIVE get SOURCE OUTPUT

get kopiert den Inhalt der Datei *SOURCE* aus dem VFS *ARCHIVE* und speichert sie unter *OUTPUT* im Host-Dateisystem.

Diese Operation benötigt folgende Parameter:

Parameter	**Beschreibung**
SOURCE	Die Datei, die aus dem VFS kopiert werden soll.
OUTPUT	Die Datei im Host-Dateisystem, in die der Inhalt von SOURCE geschrieben werden soll.

Der Aufruf kann folgende Werte zurückliefern:

Rückgabewert	Bedeutung
0	Das Kommando wurde fehlerfrei ausgeführt.
2	Das VFS konnte nicht geföffnet werden.
21	Die Datei existiert nicht im VFS.
30	Die Zieldatei im Host-Dateisystem konnte nicht erstellt werden.

Beispiel für den Aufruf:

```
./vfs /home/student/archiv get Dateiname /home/student/destination
```

Versucht die Datei *Dateiname* aus dem VFS */home/student/archiv* in die Datei */home/student/destination* im Hostdateisystem zu kopieren.
Die Datei wird dabei nicht aus dem VFS gelöscht.

3.4 ./vfs ARCHIVE del TARGET

del löscht eine Datei im VFS.

Diese Operation benötigt folgende Parameter:

Parameter	Beschreibung
TARGET	Die Datei, die im VFS gelöscht werden soll.

Der Aufruf kann folgende Werte zurückliefern:

Rückgabewert	Bedeutung
0	Das Kommando wurde fehlerfrei ausgeführt.
2	Das VFS konnte nicht geföffnet werden.
21	Die Datei existiert nicht im VFS.

Beispiel für den Aufruf:

```
./vfs /home/student/archiv del Dateiname
```

Versucht die Datei *Dateiname* aus dem VFS (*/home/student/archiv*) zu löschen.

3.5 ./vfs ARCHIVE free

free liefert die Anzahl der freien Bytes des VFS *ARCHIVE*.

Diese Operation benötigt keine eigenen Parameter.

Der Aufruf kann folgende Werte zurückliefern:

Rückgabewert	Bedeutung
0	Das Kommando wurde fehlerfrei ausgeführt.
2	Das VFS konnte nicht geföffnet werden.

Beispiel für den Aufruf:

```
./vfs /home/student/archiv free
```

Als Ausgabe wird eine Ganzzahl erwartet, die die freien Bytes des VFS angibt.

3.6 ./vfs ARCHIVE used

-used liefert die Anzahl der zum Speichern der Dateien benötigten Bytes des VFS *ARCHIVE*. Es gilt

$$BLOCKSIZE \cdot BLOCKCOUNT = free + used$$

Diese Operation benötigt keine eigenen Parameter.

Der Aufruf kann folgende Werte zurückliefern:

Rückgabewert	Bedeutung
0	Das Kommando wurde fehlerfrei ausgeführt.
2	Das VFS konnte nicht geföffnet werden.

Beispiel für den Aufruf:

```
./vfs /home/student/archiv used
```

Als Ausgabe wird eine Ganzzahl erwartet, die die belegten Bytes des VFS angibt.

3.7 ./vfs ARCHIVE list

list listet die im VFS gespeicherten Dateien auf.

Diese Operation benötigt keine eigenen Parameter.

Der Aufruf kann folgende Werte zurückliefern:

Rückgabewert	Bedeutung
0	Das Kommando wurde fehlerfrei ausgeführt.
2	Das VFS konnte nicht geföffnet werden.

Beispiel für den Aufruf:

```
./vfs /home/student/archiv list
```

Eine Beispielausgabe könnte wie folgt aussehen:

```
Dateiname1,1000,1,0
Dateiname2,9200,9,1,2,4,5,3,9,8,7,6
```

Pro Zeile werden durch Kommata getrennt folgende Werte erwartet:

- Dateiname
- Dateilänge (die wirkliche Länge der Datei, nicht der belegte Platz im VFS)
- Anzahl der durch diese Datei belegten Blöcke im Dateisystem
- Die Liste der im VFS durch diese Datei belegten Blöcke. Wie in der Informatik üblich beginnt die Nummerierung der Blöcke bei 0.

3.8 ./vfs ARCHIVE defrag

Durch wiederholtes Löschen und neu Anlegen von Dateien kann es im VFS zu Fragementierung kommen. **defrag** sorgt dafür, dass das VFS defragmentiert wird. Nachdem defrag durchgeführt wurde muss *ARCHIVE.store* folgende Bedingungen erfüllen:

- Alle belegten Blöcke befinden sich am Anfang der *ARCHIVE.store* Datei, alle freien Blöcke als ein großer Bereich am Ende.
- Alle Blöcke einer Datei im VFS liegen direkt hintereinander (in aufsteigender Reihenfolge).

In welcher Reihenfolge die Dateien abgelegt werden spielt keine Rolle.

Ist beispielsweise die Ausgabe von list vor der Defragmentierung wie folgt:

```
./vfs /home/student/archiv list
Dateiname1,1200,2,10
Dateiname2,9200,9,1,2,4,5,3,9,8,7,6
```

so könnte list nach dem Aufruf von defrag die folgende Ausgabe liefern:

```
./vfs /home/student/archiv list
Dateiname1,1200,2,0,1
Dateiname2,9200,9,2,3,4,5,6,7,8,9,10
```

Diese Operation benötigt keine Parameter.

Der Aufruf kann folgende Werte zurückliefern:

Rückgabewert	Bedeutung
0	Das Kommando wurde fehlerfrei ausgeführt.
2	Das VFS konnte nicht geföffnet werden.

Beispiel für den Aufruf:

```
./vfs /home/student/archiv defrag
```

Defragmentiert das VFS (*/home/student/archiv*).

4 Wichtige Anforderungen

Beachten Sie bei der Erstellung Ihres Programms **unbedingt** folgende Punkte:

1. Schreiben Sie Ihr Programm ausschließlich in der Programmiersprache C (und nicht in C++).
2. Ihr Programm muss unter Linux x86 lauffähig und compilierbar sein.
3. Überprüfen Sie die Funktionalität Ihres Programms (alle Operationen müssen fehlerfrei unterstützt werden).
4. Kommentieren Sie Ihren Quelltext in angemessenem Umfang.

5. Halten Sie sich strikt an das oben definierte Ein- und Ausgabeformat und die Aufrufkonventionen. Wir überprüfen Ihr Programm halbautomatisch auf die Korrektheit der ausgegebenen Lösungen. *Eine falsch formatierte Ausgabe wird nicht als korrekt anerkannt!* Im Zweifelsfall fragen Sie *rechtzeitig vor Abgabe* nach!

6. Prüfen Sie bei der Datei-Ein- und Ausgabe stets den Rückgabewert der I/O-Funktionen.

7. Geben Sie im Fehlerfall eine aussagekräftige Fehlermeldung Ihrer Wahl aus und beenden Sie das Programm mit dem passenden, oben beschriebenen *Exit Code* (*Rückgabewert*).

8. Das Programm soll sich nach Abarbeitung der Operation sofort beenden. Danach müssen alle Informationen des VFS in *vfs.store* bzw. *vfs.structure* konsistent gespeichert sein.

9. Ihr Programm muss auf der Kommandozeile ohne grafische Oberfläche lauffähig sein und darf keine interaktiven Elemente (z. B. Warten auf Benutzereingaben) enthalten.

10. Sie dürfen die ANSI-C-Standardbibliothek C89/C90 (`assert.h`, `ctype.h`, `errno.h`, `float.h`, `limits.h`, `locale.h`, `math.h`, `setjmp.h`, `signal.h`, `stdarg.h`, `stddef.h`, `stdio.h`, `stdlib.h`, `string.h`, `time.h`) sowie die Header `inttypes.h` und `stdbool.h` aus dem C99-Standard verwenden.

11. Ihr Dateisystem darf in keinen weiteren Dateien als *vfs.store* und *vfs.structure* Dateiinhalte oder Strukturdaten des VFS speichern. Einzige Ausnahme ist der get-Befehl, der natürlich in die Datei OUTPUT schreiben darf.

12. **Verwenden Sie eine einzige C-Quelltextdatei mit dem Namen vfs.c**. Ihr Programm muss sich mit einem einzelnen GCC-4.6-Compileraufruf fehlerfrei übersetzen und linken lassen. Sie können selbstverständlich auch Visual C++ (im C-Modus) oder einen anderen C-Compiler einsetzen, müssen jedoch die fehlerfreie Kompilierung unter GCC 4.6 sicherstellen. Die Verwendung einer grafischen Entwicklungsumgebung wie z. B. Visual Studio oder Eclipse kann Ihnen möglicherweise die Programmierung und vor allem das Debugging erleichtern. Sie müssen aber verstanden haben, wie die Kompilierung durch direktes Aufrufen des Compilers auf der Konsole erfolgt.

13. Legen Sie Ihr Programm so aus, dass es mit beliebig großen Eingabedaten umgehen kann. Setzen Sie hierfür unbedingt dynamische Speicherverwaltung mit `malloc/free`, gegebenenfalls auch `realloc`, ein. Wir werden Ihr Programm mit *großen* Datensätzen testen! Das bedeutet, dass (beliebig) lange Ein- und Ausgabedateien auftreten werden.

14. Ihr abgegebenes Programm wird automatisch auf Speicherlecks überprüft. Stellen Sie sicher, dass es keine Speicherlecks aufweist. Geben Sie jeglichen dynamisch angeforderten Speicher vor der Terminierung des Programms korrekt wieder frei. Für Ihre Suche nach Speicherlecks eignet sich das Programm *valgrind*[2]. Nützliche Parameter sind `--leak-check=full`, `--track-fds=yes` und `--show-reachable=yes`.

[2] http://www.valgrind.org