Abgabefrist Übungsaufgabe 5

Die Lösung muss abhängig von der Tafelübung abgegeben werden, an der ihr teilnehmt:

- **W1** – eigene Übung U5 am 20.06.-22.06.: Abgabe bis **Donnerstag, den 30.06.2022 12:00**
- **W2** – eigene Übung U5 am 27.06.-29.06.: Abgabe bis **Montag, den 04.07.2022 12:00**

In darauffolgenden Tafelübungen werden teilweise einzelne abgegebene Lösungen besprochen, teilweise auch ein Lösungsvorschlag aus dem Tutorenteam.

Allgemeine Hinweise zu den BS-Übungen

- Ab jetzt ist es *nicht* mehr möglich, Einzelabgaben im AsSESS zu tätigen. Falls ihr (statt in einer Dreiergruppe) zu zweit oder zu viert abgeben möchtet, klärt dies bitte *vorher* mit eurem Übungsleiter! Der Lösungsweg und die Programmierung sind gemeinsam zu erarbeiten.
- Die abgegebenen Antworten/Programme werden automatisch auf Ähnlichkeit mit anderen Abgaben überprüft. Wer beim Abschreiben[1] erwischt wird, verliert ohne weitere Vorwarnung die Möglichkeit zum Erwerb der Studienleistung in diesem Semester!
- Die Zusatzaufgaben sind ein Stück schwerer als die „normalen“ Aufgaben und geben zusätzliche Punkte.
- Die Aufgaben sind über AsSESS (`https://sys-sideshow.cs.tu-dortmund.de/ASSESS/`) abzugeben. Dort gibt **ein** Gruppenmitglied die erforderlichen Dateien ab und nennt dabei die anderen beteiligten Gruppenmitglieder (Matrikelnummer, Vor- und Nachname erforderlich!). Namen und Anzahl der abzugebenden C-Quellcodedateien[2] variieren und stehen in der jeweiligen Aufgabenstellung; Theoriefragen sind grundsätzlich in der Datei `antworten.txt`[3] zu beantworten. Bis zum Abgabetermin kann eine Aufgabe beliebig oft abgegeben werden – es gilt die letzte, vor dem Abgabetermin vorgenommene Abgabe.
- Sobald eine Abgabe korrigiert wurde, kann das Ergebnis ebenfalls im AsSESS eingesehen werden.

[1] Da wir im Regelfall nicht unterscheiden können, wer von wem abgeschrieben hat, gilt das für Original **und** Plagiat.
[2] codiert in UTF-8
[3] reine Textdatei, codiert in UTF-8

Aufgabe 5: Dateioperationen (10 Punkte)

In dieser Aufgabe wird der Umgang mit dem Dateisystem geübt.

ACHTUNG: Zu dieser Aufgabe existiert eine Vorgabe in Form von C-Dateien mit einem vorimplementierten Code-Rumpf, die ihr in der Programmieraufgabe erweitern sollt. Diese Vorgaben sind in Moodle[4] herunterzuladen, zu entpacken und zu vervollständigen! Die Datei `vorgaben-A5.tar.gz` lässt sich unter Linux/UNIX mittels `tar -xzvf vorgaben-A5.tar.gz` auspacken.

Theoriefragen (2 Punkte)

1. Verzeichnisse (1 Punkte)

Welche Bedeutung haben die Verzeichnisse `"."` und `".."` in einem UNIX-Dateisystem?

2. Peripheriegeräte (1 Punkt)

Beschreibt mit eigenen Worten den Unterschied zwischen block- und zeichenorientierten Geräten.

`⇒antworten.txt`

Programmieraufgabe: ID3-Tag-Leser (8 Punkte)

Beachtet, dass ihr für alle getätigten Aufrufe in euren Programmen eine entsprechende Fehlerbehandlung vorseht!

Die Implementierung soll in mehreren Schritten erfolgen, die in separaten Dateien abgegeben werden (**mp3A.c**, **mp3B.c**, [**mp3C.c**]).

a) Informationen aus Dateien extrahieren (5 Punkte)

ACHTUNG: Seid vorsichtig, wenn ihr eure Lösung auf die eigene MP3-Sammlung loslasst – bei fehlerhafter Implementierung könnte diese Schaden nehmen. Benutzt im Zweifel die von uns zur Verfügung gestellten Testdateien.

Ziel dieses Aufgabenteils ist es, ein Programm zu entwickeln, mit dem es möglich ist, **ID3v1**-Tags aus MP3-Dateien zu extrahieren und auszugeben. Euer Programm könnt ihr mit dem Aufruf `make` übersetzen. Die gewünschten Dateien sollen über per Kommandozeile übergebene Strings angegeben werden. Der Aufruf von `mp3` soll folgendermaßen aufgebaut sein:

```
mp3 <dateiname> <dateiname> ...
```

Beispiel: `$ ./mp3 /home/user/music/test.mp3 test2.mp3`

Werden dem Programm nicht ausreichend Parameter übergeben, gibt das Programm eine entsprechende Meldung aus und beendet sich. Ansonsten ist der Ablauf folgender:

1. Prüft zu Beginn, ob der angegebene Dateiname zu einer normalen Datei und nicht z. B. zu einem Verzeichnis gehört. Mit **stat(2)** könnt ihr die Eigenschaften der zu dem Pfad gehörenden Datei abfragen und mit dem Makro `S_ISREG(st_mode)` (siehe **stat(2)**-Manpage) überprüfen, ob es sich um eine normale Datei handelt. Ist dies nicht der Fall, brecht die Bearbeitung der Datei hier ab und fahrt mit der nächsten Datei fort.
2. Öffnet die übergebene Datei. Ihr könnt dazu sowohl die Funktion **open(2)** als auch die Bibliotheksfunktion **fopen(3)** verwenden.

[4] `https://sys-sideshow.cs.tu-dortmund.de/Teaching/SS2022/BS/Material`

3. Setzt den Lesezeiger mittels **lseek(2)** oder **fseek(3)** an die entsprechende Position der MP3-Datei (die letzten 128 Bytes) und lest von dieser Position bis zum Dateiende alle Bytes aus (**read(2)**/**fread(3)**).
4. Extrahiert die entsprechenden Informationen gemäß der ID3v1-Spezifikation[5]. Beachtet hierbei, dass in den ersten 3 Bytes die Kennung „TAG“ den ID3v1-Blocks einleitet. Falls keine Kennung vorhanden ist, dann brecht an dieser Stelle ab.
5. Speichert die Informationen in dem vorgegebenen `struct mp3file` und gebt die Informationen anschließend über die Konsole aus. Zur Übersetzung des Genres in einen lesbaren Text findet ihr in der Vorgabe die Funktion `const char *translateGenre(int genre_id)`.

⇒ `mp3a.c`

b) Suchen in Verzeichnissen (3 Punkte)

In diesem zweiten Aufgabenteil soll nun das in **a)** implementierte Auslesen von Informationen auf Verzeichnisse angewendet werden können.

Beispiel: `$ ./mp3 /home/user/music`

In diesem Beispiel wird das Verzeichnis `/home/user/music` nach MP3-Dateien durchsucht. Das Filtern von MP3-Dateien soll anhand der Dateiendung **.mp3** realisiert werden. Für **jede** Datei soll wieder der ID3v1-Tag ausgegeben werden. Dabei sind die Verzeichnisse mit den Namen `"."` und `".."` auszuschließen (in der Vorgabe schon implementiert). Ist die Suche in einem Verzeichnis beendet, wird der Zugriff auf das Verzeichnis mit **closedir(3)** abgeschlossen.

⇒ `mp3b.c`

c) Ersetzen in Dateien (2 Zusatzpunkte)

Ziel dieser optionalen Zusatzaufgabe ist es, Veränderungen an den MP3-Tags vorzunehmen. Übernehmt hierzu eure Lösung aus Aufgabenteil a) in die Datei `mp3c.c`. Euer Programm könnt ihr mit dem Aufruf `make mp3c` übersetzen.

Zunächst soll der Benutzer aufgefordert werden einen String einzugeben. Anschließend soll der eingelesene String in die Kommentarfelder der übergebenen Dateien geschrieben (**write(2)** bzw. **fwrite(3)**) werden. Achtet darauf, wie ihr die zu aktualisierenden Dateien öffnet (**open(2)** bzw. **fopen(3)**).

Erweitert die **main.c** um einen weiteren Parameter `--modify`, so dass der Benutzer im Voraus entscheiden kann, ob er Veränderungen an seinen MP3-Daten vornehmen möchte. Die Änderung soll auf alle übergebenen Dateien bzw. auf die übergebenen Verzeichnisse angewendet werden. Zur Übergabe des Kommentars könnt ihr den bereits vorhandenen Funktionsparameter `char *comment` nutzen.

⇒ `mp3c.c`

[5] Falls euch dazu die Informationen auf den Übungsfolien nicht ausreichen, könnt ihr die Spezifikation unter http://www.id3.org selbst nachlesen.

Tipps zu den Programmieraufgaben:

- Kommentiert euren Quellcode ausführlich, so dass wir auch bei Programmierfehlern im Zweifelsfall noch Punkte vergeben können!
- Denkt daran, dass viele Systemaufrufe fehlschlagen können! Fangt diese Fehlerfälle ab (die Aufrufe melden dies über bestimmte Rückgabewerte, siehe die jeweiligen man-Pages), gebt geeignete Fehlermeldungen aus (z.B. unter Zuhilfenahme von **perror(3)**) und beendet euer Programm danach ordnungsgemäß.
- Die Programme sollen sich mit dem gcc auf den Linux-Rechnern im IRB-Pool übersetzen lassen. Der Compiler ist mit den folgenden Parametern aufzurufen:
  `gcc -Wall -D_GNU_SOURCE`
  Weitere (nicht zwingend zu verwendende) nützliche Compilerflags sind: `-std=c11 -Wpedantic -Werror -D_POSIX_SOURCE`
- Achtet darauf, dass sich der Programmcode ohne Warnungen übersetzen lässt, z.B. durch Nutzung von `-Werror`.
- Alternativ kann auch der GNU C++-Compiler (*g++*) verwendet werden.