UNIVERSITÄT OSNABRÜCK

ESS

marcel.koeppen [☺] uni-osnabrueck.de
marcel.luetkedreimann [☺] uni-osnabrueck.de

Übung zur Vorlesung
Betriebssysteme
Wintersemester 2021/22

Programmierblatt 4: BSArchiv

➪ **Abgabe der Lösungen bis Montag, 10. Januar 14:00 im AsSESS**

Geben Sie Ihre Entwicklungsschritte jeweils in den angegebenen Dateien ab. Zu diesem Programmierblatt finden Sie im StudIP eine Vorgabe-Datei mit einer Listen-Implementierung.

In dieser Programmieraufgabe sollen Sie ein Programm entwickeln, mit dem die Dateien eines Verzeichnisbaums in einer Datei archiviert werden können. Die Daten sollen dabei allerdings nicht komprimiert werden – das überlassen wir Programmen wie gzip, bzip2 oder xz.

1 Dateiliste erstellen (7 Punkte)

Zunächst müssen die Dateien ermittelt werden, die im Archiv gespeichert werden sollen. Dazu werden ausgehend von einem Verzeichnis, das als Kommandozeilenparameter angegeben wird, die Pfade aller regulären Dateien und die zugehörigen Dateigrößen gesammelt.

1. Erstellen Sie ein Hauptprogramm, das einen Verzeichnisnamen zum Archivieren als Kommandozeilenparameter annimmt. Dabei sollen Verzeichnisse, die Pfadkomponenten enthalten, die aus einem oder zwei Punkten bestehen (also das aktuelle Verzeichnis oder einen Wechsel in eine höhere Verzeichnisebene), zurückgewiesen werden. Ein Beispiel für ein verbotenen Pfad wäre: „../das/ist/../ein/./verbotener/pfad“.
2. Implementieren Sie eine Funktion `void traverse(char *path)`, die für alle regulären Dateien im Verzeichnis `path` die Pfade und Dateigrößen ausgibt. Verwenden Sie dazu z.B. **opendir(3)**/**readdir(3)**/**closedir(3)** und **stat(2)**. Erweitern Sie `traverse` so, dass Unterverzeichnisse rekursiv durchsucht werden.
3. Die Pfade und Größenangaben der Dateien benötigen wir später mehrmals. Deshalb sollen sie in einer Liste abgespeichert werden. Dazu sind in der Vorgabe ein Listentyp **fileInfoList** und ein Elementtyp **fileInfo** definiert. Legen Sie für jede Datei ein fileInfo-Element auf dem Heap an und fügen es in eine globale fileInfoList mit der Funktion `void enqueue(fileInfoList *list, fileInfo *item)` ein. Zum Entfernen eines Elements aus der Liste gibt es die Funktion `fileInfo *dequeue(fileInfoList *list)`. Vor der ersten Benutzung muss eine fileInfoList mit der Funktion `void init(fileInfoList *list)` einmalig initialisiert werden. Um eine Liste zu traversieren, ohne die Elemente zu entfernen, können Sie ausgehend vom head-Pointer in der Liste die next-Pointer in den Elementen nutzen. Denken Sie daran, Elemente, die Sie auf dem Heap anlegen, später wieder freizugeben.
4. Geben Sie schließlich nach dem Abschluss der Traversierung die in der Liste gesammelten Pfade und Dateigrößen auf dem Bildschirm aus.

Hinweise:

- Unter Umständen müssen Sie Dateinamen zusammensetzen. Dazu können Sie die Funktion **asprintf(3)** nutzen, die wie sprintf eine formatierte Ausgabe in einen String schreib, dafür aber gleich einen String passender Länge auf dem Heap anlegt. Denken Sie daran, den String mit free freizugeben, wenn er nicht mehr benötigt wird.
- Mit **strdup(3)** können Sie eine Kopie eines Strings auf dem Heap anlegen. Denken Sie auch hier daran, den String später wieder mit free freizugeben.

→ `bsarchiv_a1.c`

©2021 Institut für Informatik, Arbeitsgruppe Eingebettete Softwaresysteme

2 Dateien archivieren (6 Punkte)

Nachdem nun bekannt ist, welche Dateien archiviert werden sollen, können wir das Archiv erstellen. Dazu schreiben wir zunächst einen kleinen Header mit einer Versionsnummer in die Archivdatei, gefolgt von einem Inhaltsverzeichnis. Schließlich müssen die Inhalte der einzelnen Dateien eingefügt werden.

Header
Inhaltsverzeichnis
Datei 1
Datei 2
...
Datei n

Tabelle 1: Archiv-Format

1. Erweitern Sie die main-Funktion so, dass sie den Namen einer Archivdatei als ersten Parameter und den schon bestehenden Parameter für den Verzeichnisnamen als zweiten Parameter annimmt.
2. Nach dem Erstellen der Dateiliste kann der Kopf der Archivdatei geschrieben werden. Öffnen Sie dazu die Archivdatei und schreiben Sie als Kennung für unser Archivformat als erstes den String „BSARCH“ (ohne Null-Byte), gefolgt von einer 16 Bit breiten, vorzeichenlosen Versionsnummer (uint16_t) in die Datei. Das BSARCH-Format ist noch ganz neu, also beginnen wir mit Version 1. Nach der Versionsnummer soll die Byte-Position des Inhaltsverzeichnisses innerhalb der Archivdatei stehen. Das ist ein vorzeichenloser 64-Bit-Wert (uint64_t). Das Inhaltsverzeichnis beginnt in der ersten Version des Dateiformats direkt nach dieser Positionsangabe. Zu beachten ist, dass hier eine binäre Datei erzeugt wird und somit auch alle Daten und Werte im binären Format in die Datei geschrieben werden müssen.
3. Vor den Einträgen des Inhaltsverzeichnisses steht in einem 64-Bit-Wert die Länge des Inhaltsverzeichnisses in Byte. Die Längenangabe selbst wird dabei **NICHT** mitgezählt. Die Längenangabe bezieht sich also nur auf die Einträge des Inhaltsverzeichnisses. Darauf folgt für jede Datei im Archiv einen Verzeichniseintrag. Diese Einträge sind wie folgt aufgebaut:

```
uint16_t pathLength;     // Länge des Dateipfades in Byte + Null-Byte
char path[path_length]; // Dateipfad mit abschließendem Null-Byte
uint64_t length;         // Länge der Datei
uint64_t startOffset;    // Offset der Datei ab Ende Inhaltsverzeichnis
```

   Der startOffset der ersten Datei nach dem Inhaltsverzeichnis ist also 0. Denken Sie daran, die Längenangabe des Inhaltsverzeichnisses anzupassen, sobald der Wert bekannt ist.
4. Kopieren Sie anschließend die Inhalte der einzelnen Dateien in die Archivdatei. Dazu bietet sich die Systemfunktion **sendfile(2)** an, die direkt im Kernel den Inhalt einer Datei in eine andere Datei kopieren kann.

Hinweise:

- Im Header `stdint.h` sind Ganzzahl-Typen mit fester Länge definiert. Vorzeichenbehaftete Typen mit 16, 32 und 64 Bit sind z.B. `int16_t`, `int32_t` und `int64_t`. Entsprechende vorzeichenlose Typen sind `uint16_t`, `uint32_t` und `uint64_t`.
- Die aktuelle Lese- bzw. Schreibposition in einer Datei lässt sich mit ftell(3), fseek(3) bzw. lseek(2) setzen und auch auslesen.

- Um zu überprüfen, ob der Archiv-Header und das Inhaltsverzeichnis richtig formatiert sind, kann es hilfreich sein, sich einen Hexdump des Archivs anzusehen. Dazu können Sie das Hilfsprogramm **hexdump(1)** benutzen. Eine schöne Darstellung mit Hexadezimal-Werten und ASCII-Interpretation bekommen Sie mit `hexdump -C eingabedatei`.
- In der Vorgabe ist ein Test-Archiv und ein Entpack-Programm zu finden, welches Sie für das Testen ihrer Implementation nutzen können!

→ `bsarchiv_a2.c`

Tipps zu den Programmieraufgaben:

- Kommentieren Sie Ihren Quellcode ausführlich, so dass wir auch bei Programmierfehlern im Zweifelsfall noch Punkte vergeben können!
- Denken Sie daran, dass viele Systemaufrufe fehlschlagen können! Fangen Sie diese Fehlerfälle ab (die Aufrufe melden dies über bestimmte Rückgabewerte, siehe die jeweiligen man-Pages), geben Sie geeignete Fehlermeldungen aus (z.B. unter Zuhilfenahme von **perror(3)**) und beenden Ihr Programm danach ordnungsgemäß.
- Die Programme sollen dem C11- und POSIX-Standard entsprechen sich mit dem gcc auf aktuellen Linux-Rechnern wie denen im CIP-Pool oder der BSVM übersetzen lassen. Z.B.:
  `gcc -std=c11 -Wall -Wextra -D_GNU_SOURCE -o bsarchiv vorgabe.c bsarchiv_ax.c`
  Weitere (nicht zwingend zu verwendende) nützliche Compilerflags sind: `-Wpedantic -Werror -D_POSIX_SOURCE`
- Achten Sie darauf, dass sich der Programmcode ohne Warnungen übersetzen lässt, z.B. durch Nutzung von `-Werror`.