UNIVERSITÄT OSNABRÜCK

marcel.luetkedreimann [☺] uni-osnabrueck.de

Übung zur Vorlesung
Betriebssysteme
Wintersemester 2023/24

Programmierblatt 4: Kopierverfahren und Buchstabensuppe

Programmierblatt 4: Kopierverfahren und Buchstabensuppe

➪ **Abgabe der Lösungen bis Montag, 15. Januar 16:00 im AsSESS**

Geben Sie Ihre Entwicklungsschritte jeweils in den angegebenen Dateien ab.

In dieser Programmieraufgabe sollen Sie verschiedene Verfahren implementieren, um eine Datei zu kopieren. Anschließend soll gemessen werden, wie zeitintensiv die verschiedenen Ansätze sind.

1 Kopierverfahren (50 Punkte)

Zunächst soll die Grundstruktur des Programmes erstellt werden. Das Programm soll später wie folgt gestartet werden können:

```
./copyfile <mode> <src> <dst>
```

Das *mode*-Argument soll später entscheiden, welches Kopierverfahren unser Programm verwenden wird. Insgesamt werden wir in den nächsten Aufgabenteilen 3 verschiedene Kopierverfahren implementieren:

1. Kopieren mit Zwischenspeicher
2. Kopieren mithilfe eines Speicherabbilds
3. Kopieren mithilfe von **sendfile(2)**

src ist der Pfad zu der Datei, welche kopiert werden soll und *dst* der Pfad zu der Zieldatei.

Erstellen Sie eine Datei *copyfile.c*, welche zunächst 3 leere Funktionen enthält: *copyUsingBuffer*, *copyUsingMap* und *copyUsingSystemCall*. Erstellen Sie außerdem eine Datei *main.c*, welche die Argumente entgegennimmt und anhand des *mode*-Arguments die korrekte Funktion aufruft. Das Projekt kann mit dem vorgegebenen Makefile gebaut werden.

Implementieren Sie die Funktion *copyUsingBuffer*: Als Erstes müssen Quell und Zieldatei mittels **open(2)** geöffnet werden. Wählen Sie geeignete Flags beim Öffnen, sodass von der Quelldatei nur gelesen werden kann und die Zieldatei ggf. erstellt wird und nur beschrieben werden kann. Anschließend soll ein Puffer mithilfe von **malloc(3)** allokiert werden. Die Größe einer Datei kann mithilfe von **fstat(2)** ermittelt werden. Danach soll mittels **read(2)** und **write(2)** Aufrufe dieser Puffer von der Quelldatei befüllt und in die Zieldatei geschrieben werden. Als letztes müssen beide Dateien mit **close(2)** geschlossen werden und der zuvor allokierte Puffer muss mit **free(3)** freigegeben werden.

Nun soll ein Speicherabbild in der *copyUsingMap*-Funktionen verwendet werden, um die Datei zu kopieren. Dafür müssen zunächst wie bei *copyUsingBuffer* beide Dateien geöffnet werden. Anstatt einen Buffer zu allokieren, soll nun jedoch die Quelldatei mithilfe von **mmap(2)** in den Speicher eingeblendet werden. Anschließend kann mittels **write(2)** direkt von dem eingeblendeten Speicher in die Zieldatei geschrieben werden. Zusätzlich zu dem Schließen der Dateien muss auch das Speicherabbild mit wieder freigegeben werden. Dafür kann **munmap(2)** verwendet werden.

Außerdem soll die *copyUsingSystemCall*-Funktion implementiert werden. Ähnlich wie in den anderen Funktionen müssen erneut die Dateien geöffnet werden und die Dateigröße muss ermittelt werden. Anschließend soll mithilfe eines **sendfile(2)**-Aufrufs die ganze Datei kopiert werden. **write(2)** wird in dieser Variante gar nicht genötigt. Nach dem Kopieren der Datei müssen die Dateien wieder geschlossen werden.

Als Letztes soll bei jedem der Kopierverfahren die Zeit in ms gemessen werden, die die gesamte Funktion benötigt. Zu der benötigten Zeit gehören also nicht nur das reine Kopieren der Daten, sondern auch das Allokieren von Buffer, Ermitteln der Dateigröße usw. Speichern Sie am Anfang und Ende jeder Funktion ein Zeitstempel mithilfe der **clock(3)**-Funktion. Anschließend soll die gemessene Zeit ausgegeben werden.

Hinweise:

- Denken Sie bei Fehlerbehandlung an das Freigeben bereits allokierter Ressourcen!
- Für Zeitstempel wird der Datentyp *clock_t* verwendet. Um die Anzahl vergangener Millisekunden zwischen dem Zeitstempel *start* und *end* zu bekommen kann folgender Ausdruck verwendet werden: *((end - start) / (CLOCKS_PER_SEC / 1000))*

```
→main.c copyfile.c copyfile.h
```

2 Buchstabensuppe (50 Punkte)

Ziel dieser Aufgabe ist es, mit C das H in der Suppe zu finden. Genauer gesagt, möchten wir nicht nur das H finden, sondern auch alle anderen Buchstaben. Es geht nämlich darum, die Häufigkeit der einzelnen Buchstaben von a-z (bzw. A-Z) in einer Datei zu zählen.

Hier die Details:

- Die Datei 'teller' repräsentiert unseren 2D Teller, welcher allerdings ggf. nicht bis zum Rand mit Suppe gefüllt ist. Es kann also am Anfang der Datei eine Folge von Nullen geben und nach der Buchstabensuppe erneut eine Folge von Nullen. An den Rändern der Suppe sammelt sich „LAUCH“ an, welcher genutzt werden kann, um zu erkennen wo sich die Grenzen der Suppe befinden. Der „LAUCH“ zählt nicht zu den Buchstaben der Suppe dazu.
- Die Methode zum Zählen soll in der vorgegebenen Datei 'counter.c' in der Methode 'count(const char*)' implementiert werden.
- Es dürfen weitere Methoden angelegt werden. Wichtig ist aber, dass der Aufruf der Methode 'count(const char*)' genügt, um das korrekte Ergebnis zu ermitteln.
- Der an 'count(const char*)' übergebene Parameter enthält den Namen der Datei, die gezählt werden soll. Diese müsst ihr selber öffnen und lesen.
- In der Datei counter.h ist das Feld (Array) 'int alphabet[26]' deklariert. Nach dem Aufruf der Methode 'count(const char*)' soll das Feld alphabet die korrekte Häufigkeit der Buchstaben der jeweiligen Textdatei enthalten. An der ersten Stelle (also alphabet[0]) sollen dabei die Vorkommen von 'a'/'A' gezählt werden, an der zweiten Stelle von 'b'/'B' und so weiter. Groß- und Kleinbuchstaben sollen gemeinsam gezählt werden: sowohl 'a' als auch 'A' sollen alphabet[0] um eins erhöhen. Alle anderen Zeichen sollen ignoriert werden.
- Die Methode 'count(const char*)' wird durch die beigefügte Datei main.c aufgerufen. Dabei wird die Zeit gemessen, die die Methode count(const char*) benötigt.
- Die Header-Datei 'counter.h', das Modul 'main.c' und das Makefile darf nicht verändert werden.

- Es ist nicht im Sinne der Aufgabe, die Compiler Optionen vor Ort durch z.B. besondere Anweisungen im Quellcode zu umgehen.

Bei dieser Aufgabe wird überprüft ob die Häufigkeit der einzelnen Buchstaben korrekt ermittelt wird. (15 Punkte)

Weiterhin wird ermittelt, wie lange die Methode im Durchschnitt für die Analyse bei mehreren Durchläufen mit einer großen Eingabedatei benötigt. Dabei wird die Wall-Clock verwendet. Daher sollte die Aufgabe so effizient wie möglich implementiert werden. Dabei gibt es folgende Zusatzpunkte:

1. Platz: 35

2. - 3. Platz: 30

4. - 6. Platz: 25

7. - 10., Platz: 20

13. - 16., Platz: 15

16. - 18., Platz: 10

Eure Lösungen werden mit gcc 12.3.0 auf einem Ubuntu 23.04 System mit i9-10900K Prozessor (10C/20T) und Samsung SSD 970 EVO Plus evaluiert.

`→counter.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.
- Achten Sie darauf, dass sich der Programmcode ohne Warnungen übersetzen lässt, z.B. durch Nutzung von `-Werror`.