Betriebssysteme

Dateisystem

Praktikum 8

Fachhochschule Bielefeld
Campus Minden
Studiengang Informatik

Beteiligte Personen:

Name	Matrikelnummer
Peter Dick	1050185

---

15. Juni 2016

Inhaltsverzeichnis

Aufgabe 8 - Dateisysteme

1 Aufgabenstellung

Impemetieren Sie ein C-Programm, das folgende Anforderungen erfüllt:

1. Eine Datei wird zum Lesen geöffnet; anschließend wird zuerst die zweite Hälfte und dann die erste Hälfte des Dateiinhaltes auf dem Bildschirm ausgegeben.

2. Danach wird der Inhalt der Datei in eine neue Datei kopiert, wobei der Dateiname der Quell- und der Zieldatei dem Programm als Argument übergeben werden kann.

3. Die letzten 10 Zeichen der ursprünglichen Datei werden ab der 11. Stelle der neuen Datei kopiert. Das Dateiende der neuen Datei soll jetzt nach den verschobenen Daten sein (also nach dem 21. Zeichen).

4. Der Inhalt der Datei soll auf dem Bildschirm ausgegeben werden.

Hinweise:

1. Benutzen Sie für diese Aufgabe ausschließlich die POSIX Befehle zur Dateibehandlung und zur Bildschirmausgabe.

2. Die in Frage kommenden POSIX Befehle sind z.B.: open, close, lseek, read, write, ftruncate

3. Die Standard C-Bibliotheksfunktionen wie z.B. fopen, fprintf, printf usw. dürfen in Ihren Programm ausschließlich zu Debugzwecken benutzt werden.

4. Bildschirmausgabe und Kopieren einer Datei sollen in derselben Funktion realisiert werden. Das Ziel der Operation (Standardausgabe oder Zieldatei) wird der Funktion als Parameter übergeben.

5. Es sollen keinerlei Beschränkungen über die Größe der zu kopierenden Inhalte getroffen werden.

6. Fehlerausgaben (z.B. eine Datei kann nicht geöffnet werden) sollen auf stderr erfolgen.

7. Eine Dokumention der benötigten POSIX Aufrufe finden Sie z.B. hier: http://www.opengroup.org/onlinepubs/009695399/functions/contents.html

2 Aufgabe 8.1

2.1 Vorbereitung

C-Projekt anlegen.
Makefile schreiben.

2.2 Durchführung

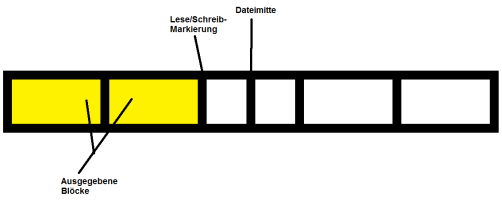
Code schreiben und dann testen bzw debuggen.

2.3 Fazit

Aufgabe 8.1.1:

```
int mycp(const int isstdout, const char *quelldatei, const char *zieldatei) {
    off_t pos = 0;
    off_t dateiMitte = 0;
    off_t dateiEnde = 0;
    char buffer[MAXBYTES] = {0};
    ssize_t count = 0;
    int datei = open(quelldatei, O_RDONLY);
    if(datei == -1) {
        char output[] = "Fehler beim oeffnen der Datei.\n";
        write(2, &output, sizeof(output));
        return 1;
    }
    dateiEnde = lseek(datei, -1L, SEEK_END);
    dateiMitte = dateiEnde / 2;
    lseek(datei, dateiMitte, SEEK_SET);
    count = read(datei, buffer, MAXBYTES);
    while(count) {
        write(1, buffer, count);
        count = read(datei, buffer, MAXBYTES);
    }
    lseek(datei, 0L, SEEK_SET);
    count = read(datei, buffer, MAXBYTES);
    pos += count;
    while(pos <= dateiMitte) {
        write(1, buffer, count);
        count = read(datei, buffer, MAXBYTES);
        pos += count;
    }
    count = dateiMitte % MAXBYTES;
    if(count) {
        write(1, buffer, count);
    }
    write(1,"\n",1);
```

Zuerst wird mit ”open(quelldatei, O_RDONLY)” die Quelldatei geöffnet. Dann wird die Position des letzten Zeichens/Bytes mit ”lseek(datei, -1L, SEEK_END)” ermittelt. Dieser Wert wird dann durch Zwei geteilt dann hat man die Dateimitte. Danach wird die Lese/Schreib-Markierung mit ”lseek(datei, dateiMitte, SEEK_SET)” zur Dateimitte verschoben. Dann wird die Datei blockweise mit ”read()” eingelesen und mit ”write(1, buffer, count);” auf dem Bildschirm ausgegeben bis das Dateiende erreicht ist. Danach wird die Lese/Schreib-Markierung mit ”lseek(datei, 0L, SEEK_SET);” wieder zum Anfang verschoben. Danach wird die Datei wieder blockweise mit ”read()” eingelesen und mit ”write(1, buffer, count);” auf dem Bildschirm ausgegeben solange die Lese/Schreib-Markierung kleiner als die Dateimitte ist. Die Position der Lese/Schreib-Markierung wird mit ”pos += count;” bei jeden Durchgang ermittelt.



Dann wird die Anzahl der restlichen Daten mit ”count = dateiMitte % MAX-BYTES;” berechnet und dann mit ”write()” ausgegeben. Dabei steht die Eins für ”stdout”.

Aufgabe 8..1.2-4 + Hinweis 4:

```
int main(int argc, char **argv) {
    char *quelldatei;
    char *zieldatei;
    int isstdout = 0;
    if(argc < 3) {
        char output[] = "Zu wenig Komandozeilenargumente.\n";
        write(2, &output, sizeof(output));
        return 1;
    } else if (argc == 3) {
        if(isStdout(argv[1])) {
            isstdout = 1;
            quelldatei = argv[2];
            zieldatei = NULL;
        } else {
            quelldatei = argv[1];
```

```
            zieldatei = argv[2];
        }
    } else if (argc == 4) {
        if(isStdout(argv[1])) {
            isstdout = 1;
        }
        quelldatei = argv[2];
        zieldatei = argv[3];
    }
    if(strCompare(quelldatei, zieldatei)) {
        char output[] = "Quell- und Zieldatei sind gleich.\n";
        write(2, &output, sizeof(output));
        return 1;
    }
    return mycp(isstdout, quelldatei, zieldatei);
}
```

Erlaubte Programmaufrufe:

- ./mycp <Quelldatei> <Zieldatei>
- ./mycp -o <Quelldatei>
- ./mycp -o <Quelldatei> <Zieldatei>
- ./mycp <Text> <Quelldatei> <Zieldatei> (<Text> wird nicht beachtet)

Der Flag "-o" legt fest das die Datei auf "stdout" ausgegeben wird. Die Quelldatei und die Zieldatei dürfen nicht gleich sein. "argc" muss mindenstens gleich Drei sein. Wenn "argc" gleich Drei ist dann wird mit der Funktion "isStdout()" geprüft ob bei "argv[1]" der "o-Flag" gesetzt ist wenn ja dann ist die Quelldatei in "argv[2]" ansonsten ist die Quelldatei in "argv[1]" und die Zieldatei in "argv[2]". Wenn "argc" gleich Vier ist dann ist die Quelldatei in "argv[2]" und die Zieldatei in "argv[3]". Auch hier wird mit der Funktion "isStdout()" geprüft ob bei "argv[1]" der "o-Flag" gesetzt ist. Danach wird mit "strCompare()" verglichen ob Quelldatei und Zieldatei gleich sind. Zuletzt werden die drei Parameter der Funktion "mycp()" übergeben.

isStdout():

```
int isStdout(const char * arg) {
    int isstdout = 0;
    int j = 1;
    for(int i = 0; i < 3; i++) {
        char c = *(arg + i);
        if(c == '-') {
            isstdout += 2 * j;
```

```
        } else if(c == 'o') {
            isstdout += 3 * j;
        } else if(c == '\0') {
        } else {
            isstdout += j;
        }
        j *= 10;
    }
    if (isstdout == 32) {
        return 1;
    }
    return 0;
}
```

Die Funktion ermittelt ob der übergebene String gleich "-o\0" ist wenn ja dann ist der rückgabe Wert Eins ansonsten Null.
Dabei wird jeden Zeichen eine Gewichtung zugeordnet:

- 0 - "\0"
- 1 - Jedes Zeichen außer "-" und "o'
- 2 - "-"
- 3 - "o"

Die Gewichte der einzelnen Zeichen werden dann zusammen gerechnet wobei vorher das Gewicht mit einen Faktor multipliziert wird. Der Faktor ist abhängig von der Position:

- 1 Stelle: Faktor = 1
- 2 Stelle: Faktor = 10
- 3 Stelle: Faktor = 100

Für "-o\0" ergibt das 32.

strCompare():

```
int strCompare(const char *str1, const char *str2) {
    char c1 = *str1;
    char c2 = *str2;
    while(c1 != '\0') {
```

```
        if(c1 != c2) {
            return 0;
        }
        c1 = *str1++;
        c2 = *str2++;
    }
    if(c1 != c2) {
        return 0;
    }
    return 1;
}
```

Die Funktion strCompare() vergleicht ob die Zwei übergebenen Strings gleich sind. Wenn ja dann wird Eins zurück gegeben ansonsten Null.
Dabei werden die Strings zeichenweise verglichen. Wenn die Zeichen nicht gleich sind wird Null zurück gegeben.

mycp():

Die Funktion mycp fürt die Dateioperationen aus. Der Parameter "isstdout" steht für den "o-Flag". Die anderen Zwei Parameter sind die Quelldatei und die Zieldatei. Bei Erfolg wird Null zurück gegeben ansonsten Eins.
Für denn ersten Teil der Funktion siehe Aufgabe 8.1.1.

```
    int dateineu;
    if(isstdout) {
        dateineu = 1;
    } else {
        dateineu = open(zieldatei, O_WRONLY|O_CREAT|O_TRUNC, S_IRWXU|S_IRWXG);
    }
    if(dateineu == -1) {
        char output[] = "Fehler beim oeffnen der Datei.\n";
        write(2, &output, sizeof(output));
        return 1;
    }
    lseek(datei, 0L, SEEK_SET);
    count = read(datei, buffer, MAXBYTES);
    while(count) {
        write(dateineu, buffer, count);
        count = read(datei, buffer, MAXBYTES);
    }
    lseek(datei, -11L, SEEK_END);
    count = read(datei, buffer, MAXBYTES);
    lseek(dateineu, -1L, SEEK_END);
    write(dateineu, buffer, count);
    close(dateineu);
    close(datei);
    return 0;
}
```

Wenn der "o-Flag" gesetzt ist wird die Variable "dateineu" gleich Eins gesetzt.

Ansonsten wird die Zieldatei mit ”open()” geöffnet.

- O_WRONLY - die Datei wird nur zum Schreiben geöffnet.
- O_CREAT - falls die Datei nicht vorhanden ist wird sie erstellt.
- O_TRUNC - der Inhalt der Datei wird gelöscht.
- S_IRWXU - setzt die rwx-Rechte für den Eigentümer.
- S_IRWXG - setzt die rwx-Rechte für die Gruppe.

Danach wird mit ”lseek()” die Lese/Schreib-Makierung zum Anfang verschoben. Dann wird mit ”read()” die Datei blockweise gelesen und dann mit ”write(dateineu, buffer, count);” in die neue Datei bzw. ”stdout” geschrieben. Das wird solange wiederholt bis das Dateiende erreicht ist. Dann wird mit ”lseek(datei, -11L, SEEK_END);” die Lese/Schreib-Markierung zum Zehnt-Letzten Zeichen verschoben. Die Zehn Zeichen werden noch mal eingelesen und dann mit ”write()” in die neue Datei geschrieben. Am ende werden beide Dateien geschlossen.

Hinweis 1-3:

Verwendet wurden die Funktionen open(), lseek(), read(), write() und close().

Hinweis 5:

Durch das blockweise lesen bzw. schreiben ist das Programm nicht von der Datei-Größe abhängig.

Hinweis 6:

Die Fehlerausgabe wird zuerst in ein char-Array gespeichert und dann mit ”write(2, &output, sizeof(output));” aus gegeben. Die Zwei steht für ”stderr”.

Hinweis 7/Quellen:

- http://pubs.opengroup.org/onlinepubs/009695399/functions/read.html Aufruf: 14.06.2016
- http://pubs.opengroup.org/onlinepubs/009695399/functions/lseek.html Auf-

ruf: 14.06.2016

- http://pubs.opengroup.org/onlinepubs/009695399/functions/write.html Aufruf: 14.06.2016

- http://pubs.opengroup.org/onlinepubs/009695399/functions/open.html Aufruf: 14.06.2016

- http://openbook.rheinwerk-verlag.de/c_von_a_bis_z/016_c_ein_ausgabe_funktionen_026.htm Aufruf: 14.06.2016