Technische Informatik 2 WS 2016/17 C05
Tutor: Marc/Bingbin

Tabea Eggers
Jan Fiedler
Florian Pflüger
Jonas Schmutte

# Übungsblatt 6

| 1 | 2 | 3 | 4 | 5 | $\sum$ |
|---|---|---|---|---|---|
| | | | | | |

Lösungsvorschlag
Abgabe: 12.12.2016

## Aufgabe 1

Unsere mycp.cc befindet sich im Ornder ouraufgabe01.
Wir lassen die Main '-1' im Fehlerfall zurüeckgeben.

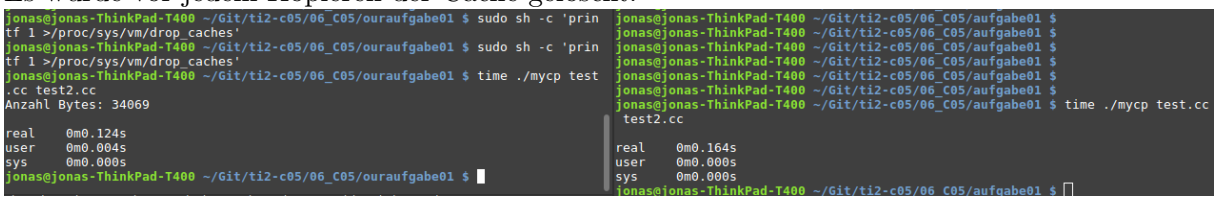
```
1  #include <iostream>
2  #include <cerrno>
3  #include <sys/stat.h>
4  #include <fcntl.h>
5  #include <unistd.h>
6  #include <sys/mman.h>
7  #include <string.h>
8
9
10 #define PAGESIZE sysconf(_SC_PAGESIZE)
11 using namespace std;
12 int main(int , char **argv){
13     //Legt Variablen an die wir brauchen
14     int offset = 0;
15     int fd_in, fd_out;
16     char* map_in, *map_out;
17     struct stat stat_in;
18     //Oeffnet einzulesendes File
19     if((fd_in = open(argv[1], O_RDONLY)) == -1){
20         perror("open in");
21         return -1;
22     }
23     //Holt Stat von eingelsesender Datei um groesse zu haben
24     if (fstat(fd_in, &stat_in) == -1){
25         perror("fstat");
26         close(fd_in);
27         return -1;
28     }
29
30
31     //Oeffnet inzuschreibende Datei
32     if((fd_out = open(argv[2], O_RDWR|O_CREAT|O_TRUNC,S_IRUSR|S_IWUSR
           )) == -1){
33         perror("open out");
```

```
34          close(fd_in);
35          return -1;
36      }
37      //Wenn eingelesene Datei leer ist, kann jetzt beendet werden.
38      if (stat_in.st_size == 0){
39          close(fd_in);
40          close(fd_out);
41          return 0;
42      }
43
44      // Macht inzuschreibende Datei beschreibbar
45      if (lseek(fd_out, stat_in.st_size-1, SEEK_SET) == -1){
46          close(fd_in);
47          close(fd_out);
48          perror("lseek");
49          return -1;
50      }
51
52      if (write(fd_out, "", 1)!= 1){
53          close(fd_in);
54          close(fd_out);
55          perror("write");
56          return -1;
57      }
58
59      if (lseek(fd_out, 0, SEEK_SET) == -1){
60          perror("lseek");
61      }
62
63      cout << "Anzahl Bytes: " << stat_in.st_size << endl;
64      //Wenn offset groesser ist als Datei sind wir fertig
65      while (offset < stat_in.st_size) {
66          // mapped eingelesene Datei in HS
67          if ((map_in = (char *) mmap(NULL, PAGESIZE, PROT_READ,
                MAP_SHARED,
68                                      fd_in, offset)) == MAP_FAILED) {
69              perror("mmap in");
70              if (close(fd_in) == -1) {
71                  perror("close in");
72              }
73              if (close(fd_out) == -1) {
74                  perror("close out");
75              }
76
77              return -1;
78          }
79          // mapped inzuschreibende Datei in HS
80          if ((map_out = (char *) mmap(NULL, PAGESIZE, PROT_READ |
81                  PROT_WRITE, MAP_SHARED, fd_out, offset)) ==
                        MAP_FAILED) {
```

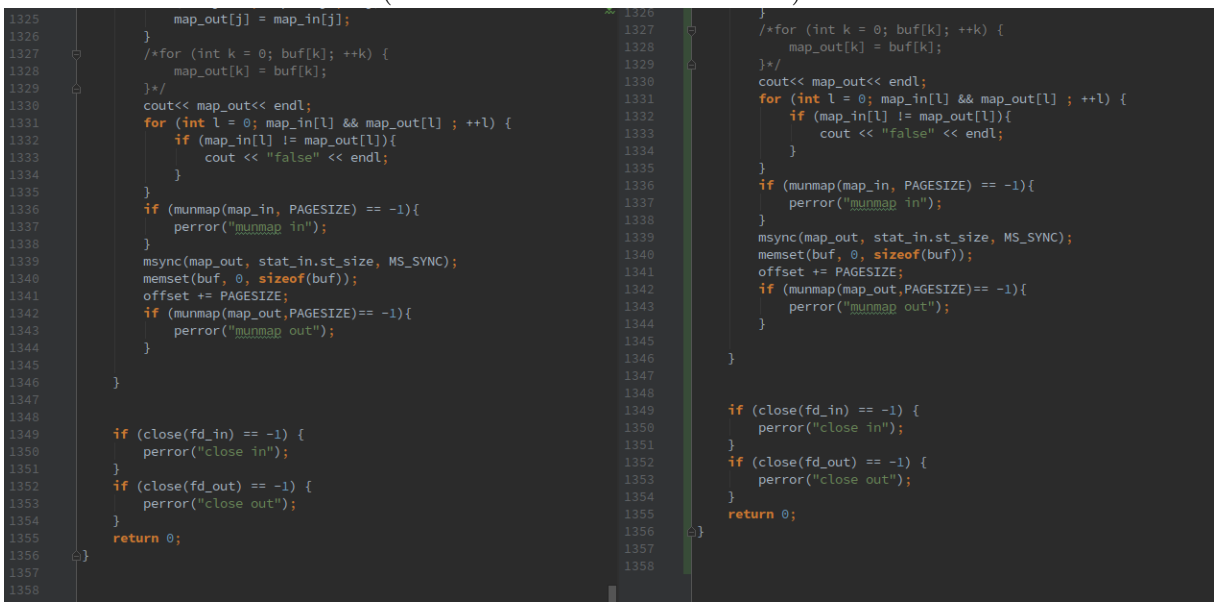
```
82              perror("mmap out");
83
84              if (munmap(map_in, PAGESIZE) == -1){
85                  perror("munmap in");
86              }
87              //nach while wird geclosed
88              break;
89          }
90          // Kopiert Inhalt der eingelesenen Datei in inzuschreibende.
91          for (int j = 0; map_in[j] && j < PAGESIZE; ++j) {
92              map_out[j] = map_in[j];
93          }
94          // eingelesene Datei kann unmapped werde.
95          if (munmap(map_in, PAGESIZE) == -1){
96              perror("munmap in");
97              //nach while wird geclosed
98              if (close(fd_in) == -1) {
99                  perror("close in");
100             }
101             if (close(fd_out) == -1) {
102                 perror("close out");
103             }
104             if (munmap(map_out, PAGESIZE) == -1){
105                 perror("munmap out");
106             }
107             return -1;
108         }
109         if (munmap(map_out,PAGESIZE)== -1){
110             perror("munmap out");
111             //nach while wird geclosed
112             if (close(fd_in) == -1) {
113                 perror("close in");
114             }
115             if (close(fd_out) == -1) {
116                 perror("close out");
117             }
118             return -1;
119         }
120         // offset muss um die Pagegroesse erhoeht werden.
121         offset += PAGESIZE;
122     }
123
124     //Dateien muesson noch geschlossen werden
125     if (close(fd_in) == -1) {
126         perror("close in");
127     }
128     if (close(fd_out) == -1) {
129         perror("close out");
130     }
131     return 0;
```

```
}
```

Ein Vergleich mit Ungerader Anzahl Bytes:
Es wurde vor jedem Kopieren der Cache gelöscht.




Das Ende der Datein danach (Der Rest stimmte auch überein):

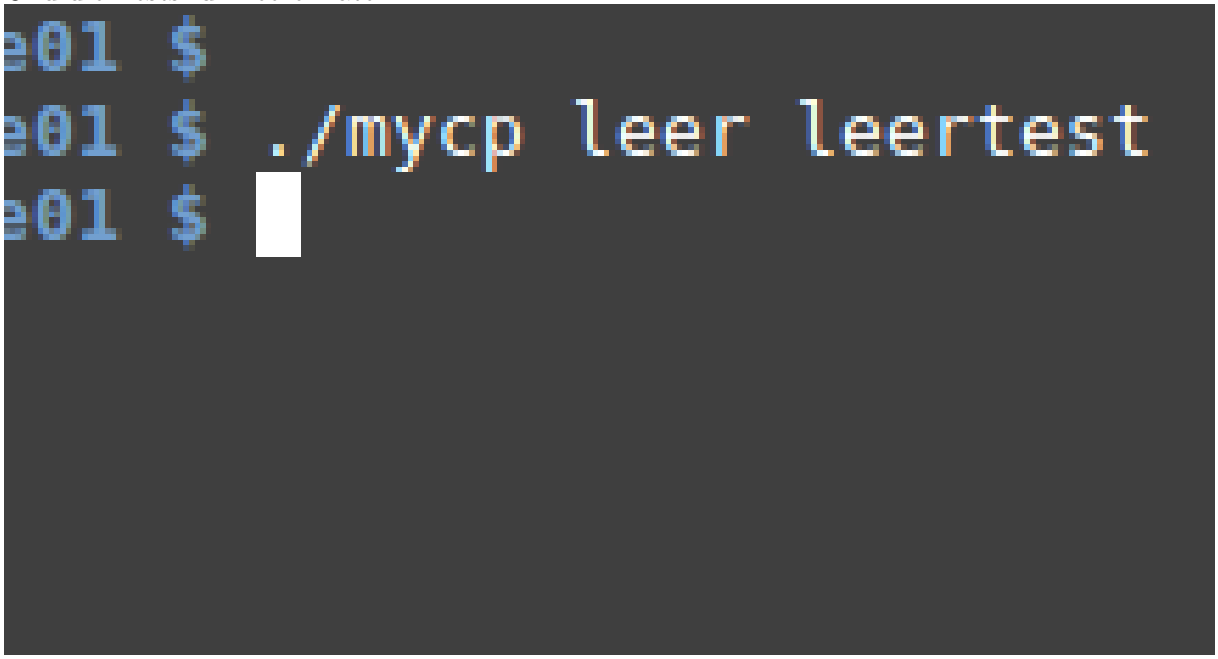


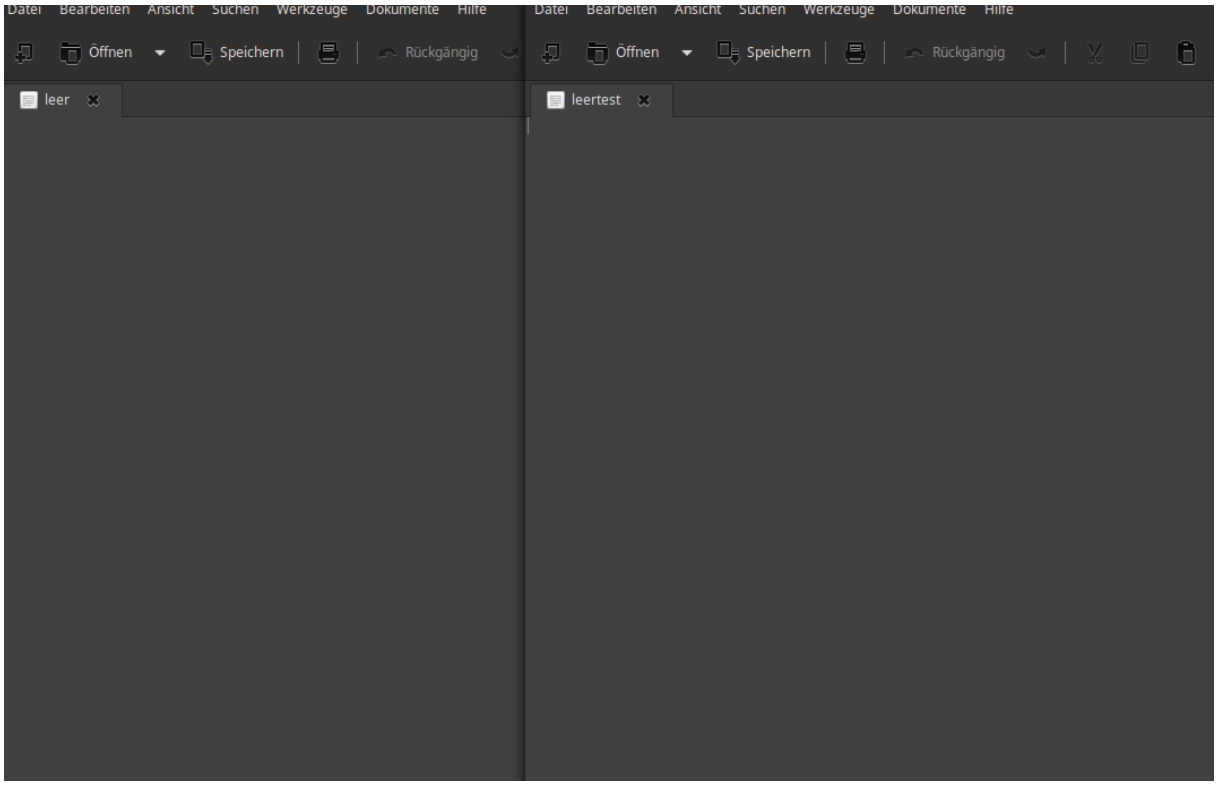
```
            map_out[j] = map_in[j];
        }
        /*for (int k = 0; buf[k]; ++k) {
            map_out[k] = buf[k];
        }*/
        cout<< map_out<< endl;
        for (int l = 0; map_in[l] && map_out[l] ; ++l) {
            if (map_in[l] != map_out[l]){
                cout << "false" << endl;
            }
        }
        if (munmap(map_in, PAGESIZE) == -1){
            perror("munmap in");
        }
        msync(map_out, stat_in.st_size, MS_SYNC);
        memset(buf, 0, sizeof(buf));
        offset += PAGESIZE;
        if (munmap(map_out,PAGESIZE)== -1){
            perror("munmap out");
        }

    }


    if (close(fd_in) == -1) {
        perror("close in");
    }
    if (close(fd_out) == -1) {
        perror("close out");
    }
    return 0;
}
```

Und die Tests für Leere Datein.






Bei uns haben auch andere Tests, das gleiche Verhalten gezeigt wie bei den hier gezeigten.

## Aufgabe 2

**gegeben:**
7200 Umdrehungen pro Minute
8 Oberflächen
100000 Spuren pro Oberfläche
1200 Sektoren pro Spur
512 Bytes pro Sektor
2,4 ms Kopfumschaltung beim Wechseln der Oberfläche
4 ms Spurenwechsel (inklusive möglicherweise erforderliche Kopfumstellung)
139586400 Byte Datei

**Berechnungen die für alle Aufgabenteile nützlich sind:**

$$\frac{139586400B}{512B} = 272629\text{Sektoren und } 352B \Rightarrow \text{insgesamt } 272630 \text{ Sektoren}$$

$$\frac{272630 \text{ Sektoren}}{1200 \text{ Sektoren pro Spur}} = 227 \text{ volle Spuren und eine mit } 230 \text{ Sektoren} \Rightarrow 228 \text{ Spuren}$$

$$7200\frac{\text{U}}{\text{min}} = 120\frac{\text{U}}{s} = 0,12\frac{\text{U}}{\text{ms}} \Rightarrow 1 \text{ U} = \frac{25}{3}\text{ms} \approx 8,333 \text{ ms}$$

**a)**
Die bestmögliche Bedingung ist: das die Zylinder immer voll belegt sind und nur der letzte nicht voll belegt ist. Somit wechselt man erst 7 mal die Oberfläche bevor man die Spur wechselt.
Berechnung:

$$\text{Die Datei braucht volle 227 Spuren und 230 Sektoren, also}$$
$$\frac{227+1}{8} = 28\text{volle Zylinder und 4Oberfläche im 9 ten Zylinder.}$$
$$(28+1)\cdot 4\text{ms} + 28\cdot 7\cdot 2,4\text{ms} + 3\cdot 2,4\text{ms} = \frac{2968}{5}\text{ms} = 593,6ms \text{ für Oberflächen und Spuren wechsel}$$
$$\text{Nun die Zeit für die Umdrehungen:}$$
$$227\cdot\frac{25}{3}\text{ms} + 230\cdot\frac{25}{3}\text{ms}\cdot\frac{1}{1200} = \frac{136315}{72}\text{ms} = 1893,26388\text{msfür alle Sektoren}$$
$$\Rightarrow \text{Datenrate} = \frac{139586560B}{\frac{2968}{5}\text{ms} + \frac{136315}{72}\text{ms}} \approx 56129,55362\frac{B}{\text{ms}}$$
$$56129,55362\frac{B}{\text{ms}}\cdot\frac{1000}{1024} \approx 54814,01721\frac{\text{KiB}}{\text{s}} \approx 53,53\frac{\text{MiB}}{\text{s}}$$

**b)**
Die schlecht möglichste Bedingung ist: nach jedem Sektor muss die Spur gewechselt werden und weil der gesuchte Sektor dann gerade vorbei ist, muss noch eine Umdrehung lang gewartet werden. Also:

$$\frac{272630\text{ Sektoren}}{100000\text{Spuren}} = 2\text{ "volle" Oberfläche und eine dritte mit 72630 Sektoren}$$
$$2\cdot 100000\cdot 4\text{ms} + 72630\cdot 4\text{ms} = 1090520\text{ms für die Sprünge}$$
$$272630\cdot\frac{25}{3}\text{ms} = 2271917,667\text{ms}$$
$$\Rightarrow \text{Datenrate:}\frac{13956560B}{1090520\text{ms} + 2271917,667\text{ms}} = \frac{1536}{37}\frac{B}{\text{ms}}$$
$$= \frac{1536}{37}\cdot\frac{1000}{1024} = \frac{1500}{37}\frac{KiB}{s} \approx 40,541\frac{KiB}{s}$$

**c)**

Gleiche Verteilung wie in b) nun aber mit 4 KiB Blöcken

$$\frac{272630Sektoren}{8} = 34079\text{ Blöcke}$$
$$34079\cdot\text{ms} = 136316\text{ms für die Sprünge}$$
$$34079\cdot\frac{25}{3}\text{ms} = \frac{851975}{3}\text{ms fürs lesen}$$
$$\Rightarrow \text{Datenrate: } = \frac{34079\cdot 8\cdot 512B}{136316\text{ms} + \frac{851975}{3}} = 332,108\frac{B}{\text{ms}} = 324,324\frac{KiB}{s}$$

c ist also etwa 280 KiB/s schneller als b

## Aufgabe 3

$10000Byte/s = 10Byte/ms$
$1s = 1000ms$
$1KiB = 1024B$

**a)**
Schreiben eines Byts dauert $0,1ms$. Dann brauchen wir inklusive auffüllen $0,1\frac{ms}{Byte} + 5ms = 5,1\frac{ms}{Byte}$. Somit hat dieses Gerät eine effektive Datenrate von

$$\frac{1000}{5,1\frac{ms}{Byte}} = 196,07843\frac{Byte}{s} \approx 196\frac{Byte}{s}$$

**b)**
Effektives schreiben eines KiB dauert

$$\frac{1024}{10\frac{Byte}{ms}} + 5ms = 107,4\frac{ms}{KiB}$$

Somit hat dieses Gerät eine effektive Datenrate von

$$\frac{1000}{107,4\frac{ms}{KiB}} \cdot 1024 = 9534,45065\frac{Byte}{s} \approx 9535\frac{Byte}{s}$$

**c)**
Wir können von der effektiven Datenrate einfach die Zeit, die noch geschrieben wird während des erneuten Auffüllens, einfach abziehen, da dies ja parallel läuft.

$$107,4\frac{ms}{KiB} - \frac{25ms}{10\frac{Byte}{ms}} = 104,9\frac{ms}{KiB}$$

Somit hat dieses Gerät eine effektive Datenrate von

$$\frac{1000}{104,9\frac{ms}{KiB}} \cdot 1024 = 9761,67779\frac{Byte}{s} \approx 9762\frac{Byte}{s}$$

**d)**
Formel umstellen:

$$\frac{1000}{x} \cdot 1024 = 10000$$
$$\frac{1000 \cdot 1024}{10000} = x = 102,4$$

Somit müssen wir die Low-Watermark so hoch setzen, das ein KiB in $102,4ms$ abgearbeitet wird.

$$107,4\frac{ms}{KiB} - 102,4\frac{ms}{KiB} = 5ms$$
$$5ms \cdot 10\frac{Byte}{ms} = 50Byte$$

Somit muss die Low-Watermark $50Byte$ betragen, damit die Netto-Datenrate des Geräts rechnerisch erreicht wird.