Technische Informatik 2 WS 2016/17 C04
Tutor: Marc Hildebrandt/ Bingbin Yu

Michael Schmidt
Stanislav Telis
Dominique Schulz
Norman Lipkow

Übungsblatt 6

1	2	3	4	5	$\sum$

Lösungsvorschlag
Abgabe: 12.12.2016

Aufgabe 1

Wir haben das Programm `mycp` mittels `make` und dem bereit gestellten Makefile übersetzt. Dabei wurden uns keine Fehler angezeigt.

```
1  dominique@dominique-Lenovo-G50-45:~/Uni/WiSe_16_17/Technische_Informatik_2/Uebungen/ueb6/
2  mycp$ make
3  g++ -Werror -Wall -Wextra -g -O2  -std=c++0x    mycp.cc   -o mycp
```

Wir haben eine neue Datei `mycp.cc` erstellt und dort die geforderten Funktionen eingebaut. Die Idee bei der Implementation war hier, dass wir die einzelnen Blöcke aus der Quelldatei in den Speicher laden. Danach wird überprüft, ob wir die Dateigröße schon abgedeckt haben (also wird die Pagegröße von der Variablen `file_size` abgezogen). Wenn ja setzen wir den Offset ab der Pagegröße (um den restlichen Inhalt der Datei zu erhalten). Also wird danach der nächste Inhalt in den Speicher geladen. Dies wird solange wiederholt bis die Variable `file_size` negativ oder 0 ist. Außerdem haben wir zur besseren Übersicht eine Hilfsfunktion eingebaut, welche die File Deskriptoren schließt, da wir den `close()` Systemaufruf in dem Code sehr oft verwenden und es nicht Übersichtlich ist, wenn man immer Fehlerbehandlung betreiben muss.

```
1  #include <iostream>
2  #include <cerrno>
3  #include <unistd.h>
4  #include <sys/mman.h>
5  #include <sys/types.h>
6  #include <sys/stat.h>
7  #include <fcntl.h>
8
9  void close_fd(int fd) {
10   if ( close(fd) == -1 ) {
11     perror("closing file descriptor failed");
12     exit(1);
13   }
14 }
15
16 int main(int argc, char **argv) {
17   //Get the system page size
18   int page_size = sysconf(_SC_PAGESIZE);
19
20   //needed for mmap
21   char *buf = NULL;
22
```

```
23    //stat struct (needed for file size)
24    struct stat source;
25
26    if ( argc > 3 ) {
27      std::cout << "Wrong Input" << std::endl;
28      std::cout << "Usage: mycp sourceFile destinationFile" << std::endl;
29      exit(1);
30    }
31
32    //Open the source file
33    int source_file;
34    if ( (source_file = open(argv[1], O_RDONLY)) == -1 ) {
35      perror("open the source file failed");
36      exit(1);
37    }
38
39    //stat the source file (we need the file size for mapping)
40    if ( fstat(source_file, &source) == -1 ) {
41      perror("fstat failed");
42      close_fd(source_file);
43      exit(1);
44    }
45
46    //Open the destination file
47    int destination_file;
48    if ( (destination_file = open(argv[2], O_RDWR | O_CREAT | O_TRUNC, S_IRUSR |
49                                  S_IWUSR)) == -1) {
50      perror("open the destination file failed");
51      close_fd(source_file);
52      exit(1);
53    }
54
55    //Get the file size
56    int file_size = source.st_size;
57
58    int bytes_written = 0;
59
60
61    //store the offset (where mapping should begin from source file)
62    int offset = 0;
63
64    //Repeat until we've mapped the whole file
65    while ( file_size > 0 ) {
66
67      //Map the file into the  memory
68      buf = (char *)mmap(NULL, (size_t)page_size, PROT_READ | PROT_WRITE, MAP_PRIVATE,
69                         source_file, offset);
70
71      //Error handling
72      if ( buf == MAP_FAILED ) {
```

```
73          perror("mmap failed");
74          close_fd(destination_file);
75          close_fd(source_file);
76          exit(1);
77        }
78
79        //write the block of bytes mapped in buf
80        bytes_written = write(destination_file, buf, page_size);
81
82        if ( bytes_written == -1 ) {
83          perror("writing to destination file failed");
84          close_fd(destination_file);
85          close_fd(source_file);
86          exit(1);
87        }
88
89      //substract the block size from the file size (cause we wrote them already)
90      file_size = file_size - page_size;
91
92      //add block size to the offset, so we get the remaining bytes
93      offset = offset + page_size;
94
95      }
96
97      //Close file descriptors
98      close_fd(destination_file);
99      close_fd(source_file);
100
101     //Unmap the memory
102     if ( munmap(buf, page_size) == -1 ) {
103       perror("munmap failed");
104       exit(1);
105     }
106
107     return 0;
108   }
```

Tests

Wir beginnen in dem wir das Makefile mittels unserer Implementierung und er naiven Implementierung vergleichen.

Unsere Implementierung:

```
1  dominique@dominique-Lenovo-G50-45:~/Uni/WiSe_16_17/Technische_Informatik_2/Uebungen/ueb6
2  /mycp$ sudo sh -c "printf 1 >/proc/sys/vm/drop_caches"
3  [sudo] Passwort für dominique:
4  dominique@dominique-Lenovo-G50-45:~/Uni/WiSe_16_17/Technische_Informatik_2/Uebungen/ueb6
5  /mycp$ time ./mycp Makefile testMake
6
7  real    0m0.165s
```

```
8   user    0m0.000s
9   sys     0m0.004s
10  dominique@dominique-Lenovo-G50-45:~/Uni/WiSe_16_17/Technische_Informatik_2/Uebungen/ueb6
11  /mycp$ cat testMake
12  LINK.o = $(LINK.cc)
13  CFLAGS=-Werror -Wall -Wextra -g -O2 #-pg
14  CXXFLAGS=-Werror -Wall -Wextra -g -O2 #-pg
15  CXXFLAGS+=-std=c++0x
16  #CXXFLAGS+=-std=c++11
17
18  PROGRAMS:=mycp
19
20  .PHONY: all clean
21  all: $(PROGRAMS)
22
23  clean:
24          -$(RM) *~ *.o core $(PROGRAMS)
25
```

Die Referenz Implementierung:

```
1   dominique@dominique-Lenovo-G50-45:~/Uni/WiSe_16_17/Technische_Informatik_2/Uebungen/ueb6/
2   aufgabe1$ sudo sh -c "printf 1 >/proc/sys/vm/drop_caches"
3   [sudo] Passwort für dominique:
4   dominique@dominique-Lenovo-G50-45:~/Uni/WiSe_16_17/Technische_Informatik_2/Uebungen/ueb6/
5   aufgabe1$ time ./mycp Makefile testMake
6
7   real    0m0.173s
8   user    0m0.004s
9   sys     0m0.000s
10  dominique@dominique-Lenovo-G50-45:~/Uni/WiSe_16_17/Technische_Informatik_2/Uebungen/ueb6/
11  aufgabe1$ cat testMake
12  LINK.o = $(LINK.cc)
13  CFLAGS=-Werror -Wall -Wextra -g -O2 #-pg
14  CXXFLAGS=-Werror -Wall -Wextra -g -O2 #-pg
15  CXXFLAGS+=-std=c++0x
16  #CXXFLAGS+=-std=c++11
17
18  PROGRAMS:=mycp
19
20  .PHONY: all clean
21  all: $(PROGRAMS)
22
23  clean:
24          -$(RM) *~ *.o core $(PROGRAMS)
25
```

Anschließend haben wir eine Datei mit dem Namen Zeit erstellt, wo wir zwei Artikel von Zeit-Online in Dateien kopiert haben. Es handelt sich um den Artikel `http://www.zeit.de/digital/datenschutz/2016-12/ransomware-loesegeld-oder-andere-rechner-infizieren` sowie diesen Artikel
`http://www.zeit.de/politik/ausland/2016-12/us-wahl-russland-hacker-donald-trump`

An diesem Beispiel kann man gut sehen, dass unsere Implementierung doppelt so schnell arbeitet wie die naive Implementierung.

Unsere Implementierung:

```
dominique@dominique-Lenovo-G50-45:~/Uni/WiSe_16_17/Technische_Informatik_2/Uebungen/ueb6
/mycp$ sudo sh -c "printf 1 >/proc/sys/vm/drop_caches"
dominique@dominique-Lenovo-G50-45:~/Uni/WiSe_16_17/Technische_Informatik_2/Uebungen/ueb6
/mycp$ time ./mycp zeit zeitgeist

real    0m0.157s
user    0m0.004s
sys     0m0.000s
dominique@dominique-Lenovo-G50-45:~/Uni/WiSe_16_17/Technische_Informatik_2/Uebungen/ueb6
/mycp$ ls -l
insgesamt 1720
-rw-rw-r-- 1 dominique dominique      1 Dez 12 16:55 empty
-rw-rw-r-- 1 dominique dominique 819200 Dez 12 19:38 foo
-rw------- 1 dominique dominique   4096 Dez 12 16:56 leer
-rw-r--r-- 1 dominique dominique    240 Nov 29 20:59 Makefile
-rwxrwxr-x 1 dominique dominique  44464 Dez 12 19:48 mycp
-rw-rw-r-- 1 dominique dominique   2597 Dez 12 19:48 mycp.cc
-rw-rw-r-- 1 dominique dominique   2582 Dez 12 19:11 mycp.cc~
-rw-rw-r-- 1 dominique dominique      1 Dez 12 12:38 read
-rw------- 1 dominique dominique  12288 Dez 12 19:35 test
-rw------- 1 dominique dominique   4096 Dez 12 13:07 test2
-rw------- 1 dominique dominique 819200 Dez 12 20:00 testfile
-rw------- 1 dominique dominique   4096 Dez 12 20:17 testMake
-rw-rw-r-- 1 dominique dominique   8065 Dez 12 14:46 ungerade
-rw-rw-r-- 1 dominique dominique  10677 Dez 12 16:58 zeit
-rw------- 1 dominique dominique  12288 Dez 12 20:29 zeitgeist

```

Die naive Implementierung:

```
dominique@dominique-Lenovo-G50-45:~/Uni/WiSe_16_17/Technische_Informatik_2/Uebungen/ueb6/
aufgabe1$ sudo sh -c "printf 1 >/proc/sys/vm/drop_caches"
dominique@dominique-Lenovo-G50-45:~/Uni/WiSe_16_17/Technische_Informatik_2/Uebungen/ueb6/
aufgabe1$ time ./mycp zeit zeitgeist

real    0m0.311s
user    0m0.008s
sys     0m0.000s
dominique@dominique-Lenovo-G50-45:~/Uni/WiSe_16_17/Technische_Informatik_2/Uebungen/ueb6/
aufgabe1$ ls -l
insgesamt 1696
-rw-rw-r-- 1 dominique dominique 819200 Dez 12 19:38 foo
-rw-r--r-- 1 dominique dominique    240 Nov 29 20:59 Makefile
-rwxrwxr-x 1 dominique dominique  37336 Dez  6 13:58 mycp
-rw-r--r-- 1 dominique dominique    938 Nov 29 20:59 mycp.cc
-rw------- 1 dominique dominique  10677 Dez 12 19:35 test
-rw-rw-r-- 1 dominique dominique 819200 Dez 12 20:00 testfile
```

```
-rw------- 1 dominique dominique    240 Dez 12 20:17 testMake
-rw-rw-r-- 1 dominique dominique   8065 Dez 12 14:46 ungerade
-rw-rw-r-- 1 dominique dominique  10677 Dez 12 16:58 zeit
-rw------- 1 dominique dominique  10677 Dez 12 20:29 zeitgeist
```

Anschließend haben wir versucht eine leere Datei mit dem Namen `empty` zu kopieren.

Unsere Implementierung:

```
dominique@dominique-Lenovo-G50-45:~/Uni/WiSe_16_17/Technische_Informatik_2/Uebungen/ueb6
/mycp$ ls -l
insgesamt 1720
-rw-rw-r-- 1 dominique dominique      0 Dez 12 20:36 empty
-rw-rw-r-- 1 dominique dominique 819200 Dez 12 19:38 foo
-rw------- 1 dominique dominique   4096 Dez 12 16:56 leer
-rw-r--r-- 1 dominique dominique    240 Nov 29 20:59 Makefile
-rwxrwxr-x 1 dominique dominique  44464 Dez 12 19:48 mycp
-rw-rw-r-- 1 dominique dominique   2597 Dez 12 19:48 mycp.cc
...
-rw-rw-r-- 1 dominique dominique   8065 Dez 12 14:46 ungerade
-rw-rw-r-- 1 dominique dominique  10677 Dez 12 16:58 zeit
-rw------- 1 dominique dominique  12288 Dez 12 20:29 zeitgeist
dominique@dominique-Lenovo-G50-45:~/Uni/WiSe_16_17/Technische_Informatik_2/Uebungen/ueb6/
mycp$ sudo sh -c "printf 1 >/proc/sys/vm/drop_caches"
dominique@dominique-Lenovo-G50-45:~/Uni/WiSe_16_17/Technische_Informatik_2/Uebungen/ueb6/
mycp$ time ./mycp empty emptytest

real    0m0.161s
user    0m0.000s
sys     0m0.004s
dominique@dominique-Lenovo-G50-45:~/Uni/WiSe_16_17/Technische_Informatik_2/Uebungen/ueb6/
mycp$ ls -l
insgesamt 1720
-rw-rw-r-- 1 dominique dominique      0 Dez 12 20:36 empty
-rw------- 1 dominique dominique      0 Dez 12 20:38 emptytest
-rw-rw-r-- 1 dominique dominique 819200 Dez 12 19:38 foo
-rw------- 1 dominique dominique   4096 Dez 12 16:56 leer
-rw-r--r-- 1 dominique dominique    240 Nov 29 20:59 Makefile
-rwxrwxr-x 1 dominique dominique  44464 Dez 12 19:48 mycp
-rw-rw-r-- 1 dominique dominique   2597 Dez 12 19:48 mycp.cc
...
-rw------- 1 dominique dominique   4096 Dez 12 20:17 testMake
-rw-rw-r-- 1 dominique dominique   8065 Dez 12 14:46 ungerade
-rw-rw-r-- 1 dominique dominique  10677 Dez 12 16:58 zeit
-rw------- 1 dominique dominique  12288 Dez 12 20:29 zeitgeist
```

Hier testen wir, ob unsere Implementierung mit Nullbytes (\0) zurecht kommt.

Unsere Implementierung:

```
dominique@dominique-Lenovo-G50-45:~/Uni/WiSe_16_17/Technische_Informatik_2/Uebungen/ueb6
/mycp$ dd if=/dev/zero bs=1 count=1 >> null
1+0 Datensätze ein
```

```
1+0 Datensätze aus
1 byte copied, 0,000181288 s, 5,5 kB/s
dominique@dominique-Lenovo-G50-45:~/Uni/WiSe_16_17/Technische_Informatik_2/Uebungen/ueb6
/mycp$ ls -l
insgesamt 1724
...
-rwxrwxr-x 1 dominique dominique  44464 Dez 12 19:48 mycp
...
-rw-rw-r-- 1 dominique dominique      1 Dez 12 20:44 null
...
dominique@dominique-Lenovo-G50-45:~/Uni/WiSe_16_17/Technische_Informatik_2/Uebungen/ueb6
/mycp$ time ./mycp null nulltest

real    0m0.005s
user    0m0.008s
sys     0m0.000s
dominique@dominique-Lenovo-G50-45:~/Uni/WiSe_16_17/Technische_Informatik_2/Uebungen/ueb6
/mycp$ ls -l
insgesamt 1728
...
-rwxrwxr-x 1 dominique dominique  44464 Dez 12 19:48 mycp
...
-rw-rw-r-- 1 dominique dominique      1 Dez 12 20:44 null
...
-rw------- 1 dominique dominique   4096 Dez 12 20:45 nulltest
...

```

Aufgabe 2

Vorüberlegungen:
nikolaus.avi = 139.586.400 B = 136.314,844 KiB = 133,119965 MiB
Größe der Platte = 512 B * 1200 (Sektoren) * 100.000 (Spuren) * 8 (Oberflächen) = 491.520.000 KiB = 480.000 MiB = 468,75 GiB

Größe pro Spur = 1200 (Sektoren) * 512 B = 614.400 B = 600 KiB
Größe pro Oberfläche = 600 KiB * 100.000 = 60.000.000 B = 58.593.75 KiB = 57,220459 GiB

Lesegeschwindigkeit:
7200 Umdrehungen/min = 120 Umdrehungen/s = 8,33 ms für eine Umdrehung = 1 Spur

a)

Zu lesen= $\frac{136.314,844 KiB}{600 KiB}$ = 227,191407 Spuren = 227 Spuren + 230 Sektoren (da 0,644 Spur*600 KiB = 114,844 KiB / 0,5 KiB = 229,688 $\approx$ 230 Sektoren) 227 * 8,33 ms = 1.890,91 ms + 5,36590833 ms (da $\frac{8,33 ms/Spur}{1200 Sektoren}$ = 0,00694167 * 230 Sektoren = 1,5965841 ms) = 1.892,50658 ms = 1,89250658 s zum Lesen der Datei.
Raufaddiert müssen noch die 1,6 ms (4ms-2,4ms da in unserem optimalen Fall der Kopf nicht auf eine andere Oberfläche gestellt werden muss) pro Spurwechsel.

Dies sind 232 * 1,6 ms = 363,2 ms = 0,3632 s

Also: 1,89250658 s + 0,3712 s = 2,25570658 s Daraus ergibt sich eine durchschnittliche Datenrate von: $\frac{136.314,844KiB}{2,25570658s}$ = 60.431,1062 KiB/s = 59,0147522 MiB/s ≈ 60.000 KiB/s ≈ 59 MiB/s

b)

Der schlechteste Fall tritt auf wenn immer nur ein Sektor auf der Oberfläche zu Lesen ist und der nächste zu lesende Sektor sich an der untersten Oberfläche befindet. Das heißt, der Kopf muss sich nach jedem gelesene Sektor 7 Oberflächen wechseln. Das sind 7 * 4ms = 28ms nach jedem gelesenen Sektor extra. Wenn man dann davon ausgeht, dass der Kopf eine komplette Umdrehung braucht sind das 8,33 ms + 28 ms = 36,33 ms die die Platte pro Sektor braucht zum lesen. Wir wissen, dass 278.400 Sektoren (227 Spuren *1200) + 230 Sektoren = 278.630 Sektoren gelesen werden müssen. Somit würde das Lesen der Datei im schlechtesten Fall 278.630 Sektoren * 36,33 ms = 10.122.627,9 ms = 10.122,6279 s

Das wäre eine Datenrate von $\frac{136.314,844KiB}{10.122,6279s}$ = 13,4663494 KiB/s ≈ 13 KiB/s ≈ 0,01 MiB/s

c) Wenn immer ein logischer Block von 4096 B gelesen wird. Dann müssen $\frac{136.314,844KiB}{4KiB}$ = 34.078,7111 ≈ 34.079 logische Blöcke gelesen werden. Die Zeit von 36,33 ms zum Lesen der Blöcke wird beibehalten, da immer noch der schlechteste Fall eintritt, dass eine komplette Umdrehung benörtigt wird, um den Block zu lesen. Das bedeutet in diesem Szenario würde es 34.079 * 36,33 ms = 1238.090,07 ms = 1238,09007 s dauern um die Datei zu lesen.

Das ergibt eine Datenrate von $\frac{136.314,844KiB}{1238,09007s}$ = 110,100911 KiB/s ≈ 110 KiB/s ≈ 0,11 MiB/s

Aufgabe 3

a) Senden von einem Zeichen ergibt sich aus $\frac{1s}{10.000}$ = 0,1 ms
Das Aufwecken und Füllen der Warteschlange dauert 5 ms. Das bedeutet, dass alle 5,1 ms ein Byte gesendet wird.
Daraus kann abgeleitet wissen wie hoch die effektive Datenrate ist = $\frac{1000ms}{5,1ms}$ = 196,078431. D.h. die effektive Datenrate liegt bei ungefähr 196 Byte/s

b)

Wir gehen hier analog wie in a) vor:
($\frac{1s}{10000}$) * 1024 = 102,4ms (Dauer für das Senden von 1 KiB
Das Aufwecken und Füllen der Warteschlange dauert 5 ms. Das bedeutet, dass alle 107,4 ms ein KiB gesendet wird.
Daraus kann abgeleitet werden wie hoch die effektive Datenrate ist = $\frac{1000ms}{107,4ms}$ = 9,31098696. D.h. die effektive Datenrate liegt bei ungefähr (9,31098696*1024 → gerundet) 9534 Byte/s.

c)

Wir gehen hier analog wie in a) und b) vor:
(1 s / 10000) * 25 = 2,5ms (Dauer für das Senden von 25 Bytes)
Wenn die Low-Watermark (LWM) erreicht ist, werden in 2,5 ms weitere 25 Bytes versendet, während die Warteschlange neu gefüllt wird. Da dies 5ms dauert, muss 2,5ms lang pausiert werden, bevor weitere Daten versendet werden können.
Das bedeutet, dass die vollen 1 KiB versendet werden und dann immer 2,5ms gewartet wird bis die Warteschlange wieder gefüllt ist. Also 102,4ms (s. b)) + 2,5ms = 104,9 ms.
Daraus ergibt sich wieder $\frac{1000ms}{104,9ms}$ = 9,53288847. D.h. die effektive Datenrate liegt bei ungefähr (9,53288847*1024 → gerundet) 9762 Byte/s

d)

Wir rechnen rückwärts um die optimale LWM zu berechnen.
Wir wollen 10000 B/s $\rightarrow \frac{10000}{1024} = 9{,}765625$ KiB/s
In ms: 1000 / 9,765625 = 102,4 ms. Das entspricht genau der Datenrate für 1 KiB wenn danach nicht pausiert werden müsste. D.h. die LWM muss bei 50 Byte liegen. Da: (1s / 10000) * 50 = 5ms. Das ist genau das Zeitfenster, das benötigt wird, damit nicht pausiert werden muss.