Technische Informatik 2 | WS 2016/17 | C02
Tutor: Marc/Bingbin

Rene Engel
Dennis Jacob
Jan Schoneberg

Übungsblatt 6

1	2	3	$\sum$

Lösungsvorschlag
Abgabe: 12.12.2016

Aufgabe 1

In Aufgabe 1 haben wir ein Programm zum kopieren von Daten erstellt. Dieses basiert auf der gegeben Referenzimplementierung, nutzt allerdings den Befehl *mmap* um die Daten in den virtuellen Adressraum des Prozesses bzw. Hauptspeicher abzubilden. Dabei werden immer nur Blöcke der Größe einer Page abgebildet. Bei der Lösung der Aufgabe haben wir uns auch an folgenden Beispielen orientiert: `https://gist.github.com/sanmarcos/991042` und `http://openbook.rheinwerk-verlag.de/linux_unix_programmierung/Kap18B-003.htm` orientiert.

Implementierung

In Allen Fehlerfällen wird versucht alle geöffneten File Deskriptoren und Abbildungen zu löschen. Falls dies nicht gelingt wird darauf nicht weiter eingegangen. Nachdem die Fehlermeldung auf *std::err* ausgegeben wurde beendet das Programm mit der Rückgabe `-1`

Falls *mmap* aufgrund eines Fehler die Signale *SIGSEGV* oder *SIGBUS* sendet, werden diese von einem Signalhandler behandelt und das Programm mit der Rückgabe -1 beendet.

```
14 void signalHandler(int signum) {
15     //mapping can no be deleted because size of mapping is unknown.
16     //just close the File Descriptors
17     close(fd_in);
18     close(fd_out);
19
20     std::cerr<< signum << " Error in mmap" << std::endl;
21
22     exit(-1);
23 }
```

Zunächst werden die Argumente auf gültigkeit geprüft.

```
32   if (argc != 3) {
33     std::cerr << "Usage: " << argv[0] << " source destination" << std::endl;
34     return -1;
35   }
36
37   if(strcmp(argv[1],argv[2]) == 0) {
38     std::cerr << "source and destination can not be the same" << std::endl;
39     return -1;
40   }
```

Und die File Deskriptoren geöffnet:

```
42   //try to open source file
43   if ((fd_in = open(argv[1],O_RDONLY)) == -1){
44     perror("open fd_in");
45     return -1;
```

```
46    }
47
48    //try to open destination file with Read and Write Permission.
49    // if File does not exits create it
50    // if File exits overwrite it
51    if ((fd_out = open(argv[2],O_RDWR|O_CREAT|O_TRUNC, S_IRUSR|S_IWUSR)) == -1){
52      close(fd_in);
53      perror("open fd_out");
54      return -1;
55    }
```

Als nächstes wird die Größe der Quelldatei und die Page Größe ermittelt. Falls die Quelldatei keine Nutzdaten enthält, kann das Programm beendet werden, da mit dem öffnen des File Deskriptors die Datei neu angelegt bzw. geleert wurde:

```
57    //try to get size of source file
58    struct stat attr;
59    if(fstat(fd_in, &attr) == -1) {
60      close(fd_in);
61      close(fd_out);
62      perror("fstat");
63      return -1;
64    }
65
66    int in_size = attr.st_size;
67    if(in_size == 0) {
68      close(fd_in);
69      close(fd_out);
70      return 0;
71    }
72
73    //try to get size of one Page
74    long pagesize = sysconf(_SC_PAGESIZE);
75    if(pagesize == -1) {
76      close(fd_in);
77      close(fd_out);
78      perror("pagesize");
79      return -1;
80    }
```

Damit die Abbildung klappt, muss die Zieldatei die gleiche Größe haben, wie die Quelldatei. Dafür wird mit *lseek* der File Offset der Zieldatei an die vorletzte Stelle gesetzt und ein Nul Char (Ende einer Zeichenkette) angehängt.

```
82     //https://gist.github.com/sanmarcos/991042
83    // stretch the file size to the size of the (mmapped) array of char
84    // in_size -1 because an additional Nul Char will be added in the next step
85    if (lseek(fd_out, in_size-1, SEEK_SET) == -1) {
86        close(fd_out);
87        close(fd_in);
88        perror("Error calling lseek() to 'stretch' the file");
89        return -1;
90    }
91
92     //https://gist.github.com/sanmarcos/991042
93      /* Something needs to be written at the end of the file to
94     * have the file actually have the new size.
95     * Just writing an empty string at the current file position will do.
96     *
97     * Note:
98     *  - The current position in the file is at the end of the stretched
99     *    file due to the call to lseek().
```

```
100     *  - An empty string is actually a single '\0' character, so a zero-byte
101     *    will be written at the last byte of the file.
102     */
103    if (write(fd_out, "", 1) == -1) {
104        close(fd_out);
105        close(fd_in);
106        perror("Error writing last byte of the file");
107        return -1;
108    }
```

In der folgenden for Schleife wird das tatsächlich Kopieren durchgeführt:

```
110    int bytesToWrite = in_size;
111
112    //while not all Bytes have been written
113    for(int i = 0; bytesToWrite > 0; i++) {
114
115      //offset needs to be a multiple of the page size
116      int offset = pagesize*i;
117
118      if(bytesToWrite >= pagesize) {
119        bytesToWrite -= pagesize;
120      } else {
121
122        //bytesToWrite < pagesize
123        //not a whole page need to be mapped.
124        pagesize = bytesToWrite;
125
126        //bytesToWrite = 0; for loop will be left
127        bytesToWrite -= bytesToWrite;
128      }
129
130      //try to map source file
131      char *inmmap = (char*) mmap(NULL, pagesize, PROT_READ, MAP_SHARED, fd_in, offset
           );
132      if(inmmap == MAP_FAILED) {
133        close(fd_in);
134        close(fd_out);
135        perror("inmmap");
136        return -1;
137      }
138
139      //try to map destination file
140      char *outmmap  = (char*) mmap(NULL, pagesize, PROT_WRITE | PROT_READ,
           MAP_SHARED, fd_out, offset);
141      if(outmmap == MAP_FAILED) {
142        munmap(inmmap, pagesize);
143        close(fd_in);
144        close(fd_out);
145        perror("outmmap");
146        return -1;
147      }
148
149      //do the actual copy
150      //for(int j = 0; j<pagesize; j++) {
151      //  outmmap[j] =  inmmap[j];
152      //}
153      memcpy(outmmap, inmmap, pagesize);
154
155      //try to unmap source file
156      if(munmap(inmmap, pagesize) ==-1) {
157        munmap(outmmap, pagesize);
```

```
158         close(fd_in);
159         close(fd_out);
160         perror("unmmap inmmap");
161         return -1;
162       }
163
164       //try to unmap destination file
165       if(munmap(outmmap, pagesize) ==-1) {
166         munmap(inmmap, pagesize);
167         close(fd_in);
168         close(fd_out);
169         perror("unmmap outmmap");
170         return -1;
171       }
172     }
```

Dabei wird jeweils ein Bereich, der maximal so groß ist wie eine Page, mittels *mmap* auf den virtuellen Adressraum des Prozesses abgebildet. Falls die Quelldatei größer ist, als eine Page wird mit einem Offset gearbeitet, der ein vielfaches der Page Größe entspricht. Das eigentliche kopieren wird mit dem Befehl memcpy durchgeführt. Eine Fehlerfallbehandlung wird nicht durchgeführt, da die Methode memcpy nur einen Pointer auf den Zielspeicherbereich zurück gibt. Daran kann kein Fehler erkannt werden. Unser erster versuch die Daten Byteweise zu kopieren, wie es im auskommentierten Teil steht, wäre zu langsam.
Nach dem eigentlichen Kopieren wird die Abbildung wieder Entfernt.
Zum Schluss werden die File Deskriptoren geschlossen und noch nicht geschriebene Daten auf die Platte geschrieben:

```
174   //try to sync destination File Descriptor
175   if(fsync(fd_out) == -1) {
176     close(fd_in);
177     close(fd_out);
178     perror("fsync");
179     return -1;
180   }
181   //try to close source File Descriptor
182   if(close(fd_in) == -1) {
183     close(fd_out);
184     perror("close fd_in");
185     return -1;
186   }
187
188   //try to close destination File Descriptor
189   if(close(fd_out) == -1) {
190     perror("close fd_out");
191     return -1;
192   }
193
194   return 0;
195 }
```

Wenn dies erfolgreich war, beendet das Programm mit der Rückgabe 0.

Test

Es wurde ein Testskript `Test.sh` erstellt. Von diesem Skript aus gesehen liegen die eigene Lösung der Aufgabe im Unterverzeichnis *solution* und die Referenzlösung der Uni im Unterverzeichnis *uni*. Für die Tests werden einige Hilfsdateien verwendet. Da für die einige Befehle root Rechte benötigt werden, die uns auf den x-Rechner nicht zur Verfügung stehen, haben wir die Tests auf unseren privaten Computern sowie auf der TI2 Virtuellen Maschine durchgeführt. Die getesteten

Fälle stehen mit in dem Skript für alle Fehlerfälle wird eine entsprechende Ausgabe erwartet. Die Dateien werden bei den Tests die erfolgreich verlaufen sollen mittels *diff* miteinander verglichen.

Es werden dabei auch die geforderten Testfälle abgedeckt: Eine ungerade Datei wird mit dem ersten Testfall abgedeckt und auch eine leere Datei bzw. leere Bytes werden getestet.
Test.sh:

```
1  echo "-------------------------------------------------------"
2  echo "Test der korrekten Funktion:"
3  echo "----------------------------------------"
4  echo "  kleine Datei (3 Zeichen):"
5  echo "    Ausführung: solution/mycp solution/littleFile.txt solution/littleFileOut
       .txt"
6  sudo solution/mycp solution/littleFile.txt solution/littleFileOut.txt
7  echo "    Inhalt von littleFileOut.txt:"
8  sudo cat solution/littleFileOut.txt
9  echo "\n"
10 echo "Vergleiche Dateien:"
11 sudo diff solution/littleFile.txt solution/littleFileOut.txt
12 echo -e "\n----------------------------------------"
13 echo "  große Datei (1877443 Zeichen):"
14 echo "    Ausführung: solution/mycp solution/bigFile.txt solution/bigFileOut.txt"
15 sudo solution/mycp solution/bigFile.txt solution/bigFileOut.txt
16 echo "    Inhalt (Erste 2 Zeilen) von bigFileOut.txt:"
17 sudo head -2 solution/bigFileOut.txt
18 echo "Vergleiche Dateien:"
19 sudo diff solution/bigFile.txt solution/bigFileOut.txt
20 sudo rm solution/littleFileOut.txt
21 sudo rm solution/bigFileOut.txt
22 echo "-------------------------------------------------------"
23 echo "Leistungstest (2 Durchläufe):"
24 echo "----------------------------------------"
25 echo "  1. Durchlauf"
26 echo "--------------------------------"
27 echo "    Referenzimplementierung:"
28 echo "      Caches leeren und Ausführung: time uni/mycp uni/bigFile.txt uni/
       bigFileOut.txt"
29 sudo sh -c "sync; echo 3 > /proc/sys/vm/drop_caches"
30 sudo time uni/mycp uni/bigFile.txt uni/bigFileOut.txt
31 echo "      Inhalt (Erste 2 Zeilen) von bigFileOut.txt:"
32 sudo head -2 uni/bigFileOut.txt
33 sudo rm uni/bigFileOut.txt
34 echo "--------------------------------"
35 echo "    Lösung:"
36 echo "      Caches leeren und Ausführung: time solution/mycp solution/bigFile.txt
       solution/bigFileOut.txt"
37 sudo sh -c "sync; echo 3 > /proc/sys/vm/drop_caches"
38 sudo time solution/mycp solution/bigFile.txt solution/bigFileOut.txt
39 echo "      Inhalt (Erste 2 Zeilen) von bigFileOut.txt:"
40 sudo head -2 solution/bigFileOut.txt
41 echo "Vergleiche Dateien:"
42 sudo diff solution/bigFile.txt solution/bigFileOut.txt
43 sudo rm solution/bigFileOut.txt
44 echo "----------------------------------------"
45 echo "  2. Durchlauf"
46 echo "--------------------------------"
47 echo "    Referenzimplementierung:"
48 echo "      Caches leeren und Ausführung: time uni/mycp uni/bigFile.txt uni/
       bigFileOut.txt"
49 sudo sh -c "sync; echo 3 > /proc/sys/vm/drop_caches"
50 sudo time uni/mycp uni/bigFile.txt uni/bigFileOut.txt
```

```
51 echo "        Inhalt (Erste 2 Zeilen) von bigFileOut.txt:"
52 sudo head -2 uni/bigFileOut.txt
53 sudo rm uni/bigFileOut.txt
54 echo "--------------------------------"
55 echo "     Lösung:"
56 echo "        Caches leeren und Ausführung: time solution/mycp solution/bigFile.txt
       solution/bigFileOut.txt"
57 sudo sh -c "sync; echo 3 > /proc/sys/vm/drop_caches"
58 sudo time solution/mycp solution/bigFile.txt solution/bigFileOut.txt
59 echo "        Inhalt (Erste 2 Zeilen) von bigFileOut.txt:"
60 sudo head -2 solution/bigFileOut.txt
61 echo "Vergleiche Dateien:"
62 sudo diff solution/bigFile.txt solution/bigFileOut.txt
63 sudo rm solution/bigFileOut.txt
64 echo "------------------------------------------------------"
65 echo "Test: Leere Eingabedatei"
66 echo "---------------------------------------"
67 echo "  Ausführung: solution/mycp solution/emptyIn.txt solution/emptyOut.txt"
68 sudo solution/mycp solution/emptyIn.txt solution/emptyOut.txt
69 echo "Vergleiche Dateien:"
70 sudo diff solution/emptyIn.txt solution/emptyOut.txt
71 sudo rm solution/emptyOut.txt
72 echo "------------------------------------------------------"
73 echo "Test: Leere Eingabe"
74 echo "---------------------------------------"
75 echo "  Ausführung: solution/mycp"
76 sudo solution/mycp
77 echo "------------------------------------------------------"
78 echo "Test: Quelldatei = Zieldatei"
79 echo "  Ausführung: solution/mycp solution/littleFile.txt solution/littleFile.txt"
80 sudo solution/mycp solution/littleFile.txt solution/littleFile.txt
81 echo "------------------------------------------------------"
82 echo "Test: Nur Leserechte"
83 echo "  Wechsel zum Verzeichnis 'solution'"
84 cd solution
85 echo "  Ausführung: chmod 100 solution/littleFile.txt"
86 sudo chmod 000 littleFile.txt
87 echo "  Ausführung: mycp littleFile.txt littleFileOut.txt"
88 sudo ./mycp littleFile.txt littleFileOut.txt
89 sudo chmod 777 littleFile.txt
90 sudo rm littleFileOut.txt
91 echo "------------------------------------------------------"
```

Wie zu erkennen ist werden bei den Testfällen bei denen eine Datei als Ergebnis erwartet wird die Quell- und Zieldatei mittels `diff` miteinander verglichen. Wenn diff keine Ausgabe hat, sind beide Dateien identisch.
Hier die entsprechende Testausgabe:

```
1 ------------------------------------------------
2 Test der korrekten Funktion:
3 ----------------------------------------
4   kleine Datei (3 Zeichen):
5     Ausführung: solution/mycp solution/littleFile.txt solution/littleFileOut.txt
6     Inhalt von littleFileOut.txt:
7 abc\n
8 Vergleiche Dateien:
9
10 ----------------------------------------
11   große Datei (1877443 Zeichen):
12     Ausführung: solution/mycp solution/bigFile.txt solution/bigFileOut.txt
13     Inhalt (Erste 2 Zeilen) von bigFileOut.txt:
14 Blindtext-Generator Logo
```

```
Mehrsprachig - Übersetzung
Vergleiche Dateien:
-----------------------------------------------------
Leistungstest (2 Durchläufe):
-------------------------------------
  1. Durchlauf
-------------------------------
      Referenzimplementierung:
        Caches leeren und Ausführung: time uni/mycp uni/bigFile.txt uni/bigFileOut.
            txt
0.00user 0.05system 0:00.50elapsed 12%CPU (0avgtext+0avgdata 2240maxresident)k
11128inputs+3680outputs (20major+84minor)pagefaults 0swaps
        Inhalt (Erste 2 Zeilen) von bigFileOut.txt:
Blindtext-Generator Logo
Mehrsprachig - Übersetzung
-------------------------------
      Lösung:
        Caches leeren und Ausführung: time solution/mycp solution/bigFile.txt
            solution/bigFileOut.txt
0.00user 0.03system 0:00.19elapsed 21%CPU (0avgtext+0avgdata 2220maxresident)k
11304inputs+3680outputs (23major+1006minor)pagefaults 0swaps
        Inhalt (Erste 2 Zeilen) von bigFileOut.txt:
Blindtext-Generator Logo
Mehrsprachig - Übersetzung
Vergleiche Dateien:
-------------------------------------
  2. Durchlauf
-------------------------------
      Referenzimplementierung:
        Caches leeren und Ausführung: time uni/mycp uni/bigFile.txt uni/bigFileOut.
            txt
0.00user 0.06system 0:00.19elapsed 33%CPU (0avgtext+0avgdata 2160maxresident)k
11128inputs+3680outputs (20major+84minor)pagefaults 0swaps
        Inhalt (Erste 2 Zeilen) von bigFileOut.txt:
Blindtext-Generator Logo
Mehrsprachig - Übersetzung
-------------------------------
      Lösung:
        Caches leeren und Ausführung: time solution/mycp solution/bigFile.txt
            solution/bigFileOut.txt
0.00user 0.05system 0:00.20elapsed 26%CPU (0avgtext+0avgdata 2272maxresident)k
11288inputs+3680outputs (23major+1003minor)pagefaults 0swaps
        Inhalt (Erste 2 Zeilen) von bigFileOut.txt:
Blindtext-Generator Logo
Mehrsprachig - Übersetzung
Vergleiche Dateien:
-----------------------------------------------------
Test: Leere Eingabedatei
-------------------------------------
  Ausführung: solution/mycp solution/emptyIn.txt solution/emptyOut.txt
Vergleiche Dateien:
-----------------------------------------------------
Test: Leere Eingabe
-------------------------------------
  Ausführung: solution/mycp
Usage: solution/mycp source destination
-----------------------------------------------------
Test: Quelldatei = Zieldatei
  Ausführung: solution/mycp solution/littleFile.txt solution/littleFile.txt
source and destination can not be the same
-----------------------------------------------------
Test: Nur Leserechte
```

```
73   Wechsel zum Verzeichnis 'solution'
74   Ausführung: chmod 100 solution/littleFile.txt
75   Ausführung: mycp littleFile.txt littleFileOut.txt
76 ------------------------------------------------------
```

Wie anhand von `time` zu erkennen ist, benötigt unsere Implementierung in beiden Durchläufen eine ähnliche oder geringe Gesamtzeit als die Referenzimplementierung. Darüber lässt sich allerdings keine direkte Aussage über die tatsächliche Geschwindigkeit treffen, da darin möglicherweise auch die CPU Zeit anderer Prozesse enthalten ist. Allerdings benötigt unsere Implementierung auch eine geringe tatsächliche Zeit im User bzw. Kernel Mode. Dies lässt sich auch bei weiteren Durchläufen der Tests beobachten. Dieser spezielle Test wurde auf einem relativ schwachen Rechner ausgeführt. Bei Leistungsstarken System gibt es ggf. keinen sichtbaren Unterschied.

Durchlauf	tatsächliche Zeit unserer Lösung	tatsächliche Zeit Referenz
1	0.03s	0.05s
2	0.05s	0.06s

Die Ausgabe wurde mit diesem Skript in die Datei testoutput umgeleitet.

```
1 sudo ./Test.sh 2> testoutput.txt 1>&2
```

Aufgabe 2

Zuerst einige Definitionen von Variablen und ihrer Bedeutung, die ihn folgender Aufgabe zu Vereinfachung genutzt wurden:

T_{ges}	Zeit gesamt (hier für die gesamte Leseoperation)
T_{rB}	Zeit, die für das Lesen eines Datenblocks benötigt wird
T_U	Zeit einer Oberflächenumdrehung
T_{sw}	Dauer eines Spurenwechsels
T_{ow}	Dauer eines Oberflächenwechsels
S_D	Größe der Datei
S_B	Anzahl der Datenblöcke, die durch die Datei belegt sind
S_{Sp}	Anzahl der Datenblöcke pro Spur
S_O	Anzahl der Spuren pro Oberfläche
N_B	Anzahl der Datenblöcke (benötigt durch die Datei)
N_{Sp}	Anzahl der Spüren (benötigt durch die Datei)
N_O	Anzahl der Oberflächen (benötigt durch die Datei)
L_D	Durchschnittliche Lesegeschwindigkeit

a) best-case

> Wie lange würde es bei bestmöglichen Bedingungen dauern, die genannte Datei von der beschriebenen Festplatte zu lesen und welche durchschnittliche Lesegeschwindigkeit (Datenrate) würde dabei erreicht?

Dies ist der sogenannte **best-case**. Hier wird angenommen, das die Datei sowohl innerhalb der belegten Spuren sowie auf den belegten Oberflächen sequenziell abgespeichert ist. Außerdem wird angenommen, dass nach einem Spurenwechsel der logisch nächste Datenblock sofort lesbar ist und zu beginn der Lesekopf den ersten logischen Datenblock ohne Wartezeit erreichen kann.

$$
\begin{aligned}
N_B &= \left\lceil \frac{S_D}{S_B} \right\rceil &&= \left\lceil \frac{139586400\ B}{512\ B} \right\rceil &&= 272630 \\
N_{Sp} &= \left\lceil \frac{N_B}{S_{Sp}} \right\rceil &&= \left\lceil \frac{27230}{1200} \right\rceil &&= 273 \\
N_O &= \left\lceil \frac{N_{Sp}}{S_O} \right\rceil &&= \left\lceil \frac{273}{100000} \right\rceil &&= 1 \\
T_{rB} &= \frac{1}{\frac{7200}{60}} \cdot \frac{1}{1200}\ s &&= \frac{1}{144000}\ s &&= 6,9\overline{4}\ ns \\
T_{ges} &= T_{rB} \cdot N_B + N_{Sp} \cdot 4 \cdot 10^{-3}\ s \\
&= \frac{1}{144000}\ s \cdot 272630 + (273-1) \cdot 4 \cdot 10^{-3}\ s \\
&= 2,981263889\ s \\
L_D &= \frac{139586400}{\frac{1}{144000}\ s \cdot 272630 + (273-1) \cdot 4 \cdot 10^{-3}} \\
&= 46821215,83 \frac{B}{s} = 45723,844 \frac{KiB}{s} = 44,652 \frac{MiB}{s}
\end{aligned}
$$

b) worst-case (512 B Datenblockgröße)

> Wie lange würde es bei der schlechtest möglichen Verteilung der Datenblöcke auf der Festplatte dauern, bis die genannte Datei von der beschriebenen Festplatte gelesen wurde und welche durchschnittliche Lesegeschwindigkeit (Datenrate) würde dabei erreicht?

Dies ist der sogenannte **worst-case**. Hierbei wird angenommen, dass jeder Datenblock auf einer eigenen Spur liegt und nach jeden Spurwechsel eine ganze Umdrehung gewartet werden muss, bis der logisch nächste Block verfügbar ist. Außerdem liegt der nächste Block nach einem Oberfläche nicht in dem bisherigem Zylinder. Der Lesekopf startet in einem anderen Zylinder und der aktive Lesekopf befindet sich auf einer anderen Oberfläche.

$$
\begin{aligned}
N_B &= \left\lceil \frac{S_D}{S_B} \right\rceil && = \left\lceil \frac{139586400\ B}{512\ B} \right\rceil = 272630 \\
N_{Sp} &= N_B && = 272630 \\
N_O &= \left\lceil \frac{N_{Sp}}{S_O} \right\rceil && = \left\lceil \frac{272630}{100000} \right\rceil = 3 \\
T_{rB} &= \frac{1}{\frac{7200}{60}} \cdot \frac{1}{1200}\ s && = \frac{1}{144000}\ s = 6,9\overline{4}\ ns \\
T_{ges} &= T_{rB} \cdot N_B + N_{Sp} \cdot T_{sw} + N_{Sp} \cdot T_U + N_O \cdot T_{ow} \\
&= N_B \cdot (T_{rB} + T_U + T_{sw}) + N_O \cdot N_O \\
&= 272630 \cdot \left(\frac{1}{144000} + \frac{1}{120} + 4 \cdot 10^{-3} \right) s + 3 \cdot 2,4 \cdot 10^{-3}\ s \\
&= 3364,337131\ s \\
L_D &= \frac{139586400}{272630 \cdot \left(\frac{1}{144000}\ s + \frac{1}{120}\ s + 4 \cdot 10^{-3}\ s \right) + 3 \cdot 2,4 \cdot 10^{-3}\ s} \\
&= 41490,015 \frac{B}{s} = 40,518 \frac{KiB}{s} = 0,04 \frac{MiB}{s}
\end{aligned}
$$

c) worst-case (4 KiB Datenblockgröße)

> Wie ändert sich das Ergebnis für b), wenn jeweils acht hintereinander liegende Plattenblöcke zu einem logischen Block der Größe 4 KiB zusammengefasst werden (ein logischer Block wird immer komplett genutzt, selbst wenn nicht alle Plattenblöcke darin benötigt würden, um die Datei zu speichern).

Die Annahme hier ist die gleiche wie in dem vorherigem Abschnitt. Jedoch hat jeder Datenblock nun eine Größe von 4 KiB.

$$
\begin{array}{lcll}
N_B & = & \left\lceil \frac{S_D}{S_B} \right\rceil & = \left\lceil \frac{139586400\ B}{4096\ B} \right\rceil = 34079 \\
N_{Sp} & = & N_B & = 34079 \\
N_O & = & \left\lceil \frac{N_{Sp}}{S_O} \right\rceil & = \left\lceil \frac{34079}{100000} \right\rceil = 1 \\
T_{rB} & = & \frac{8}{\frac{7200}{60}} \cdot \frac{1}{1200}\ s & = \frac{1}{18000}\ s = 5,\overline{5}\ ns
\end{array}
$$

$$
\begin{array}{lcl}
T_{ges} & = & T_{rB} \cdot N_B + N_{Sp} \cdot T_{sw} + N_{Sp} \cdot T_U + N_O \cdot T_{ow} \\
& = & N_B \cdot (T_{rB} + T_U + T_{sw}) + N_O \cdot N_O \\
& = & 34079 \cdot \left(\frac{1}{18000} + \frac{1}{120} + 4 \cdot 10^{-3} \right) s + 3 \cdot 2,4 \cdot 10^{-3}\ s \\
& = & 422,208144\ s \\
L_D & = & \frac{139586400}{34079 \cdot \left(\frac{1}{18000} + \frac{1}{120} + 4 \cdot 10^{-3} \right) s + 3 \cdot 2,4 \cdot 10^{-3}\ s} \\
& = & 330610,3917 \frac{B}{s} = 322,86 \frac{KiB}{s} = 0,315 \frac{MiB}{s}
\end{array}
$$

Mit einer Blockgröße von 4 KiB erreicht man im **worst-case** eine ca. 80-fach höhere Lesegeschwindigkeit als mit 512 B.

Aufgabe 3

> Welche effektive Datenrate (in Byte/s) zum angeschlossenen Gerät ergibt sich in folgenden Fällen?

Der Ausgangszustand ist in allen drei Fällen ein voller Puffer des Gerätecontrollers. Zunächst wird der Puffer geleert. Dazu wird die **Entleerungszeit** t_{clear} mit der Größe des Puffers s_{buffer}, der **Datenrate des Gerätecontroller** $r_{controller}$ und der **Low-Wartermark** $l_{watermark}$ berechnet:

Entleerungszeit $t_{clear} = \frac{s_{buffer} - l_{watermark}}{r_{controller}}$

Damit und mit der **Auffüllzeit des Puffers** t_{fill} (hier 5 ms) wird die Zeit berechnet die für das Leeren und Auffüllen des Puffers benötigt wird, also eine **Periodenzeit** t_{period}:

$t_{period} = t_{clear} + t_{fill}$

Die tatsächliche Anzahl gesendeter Bytes entspricht immer der Puffergröße, da der Gerätecontroller ab dem Zeitpunkt des Erreichens der **Low-Watermark** weiter die restlichen Bytes an das Gerät sendet und parallel der Puffer mit neuen Bytes gefüllt wird. Die effektive Datenrate ergibt sich deshalb, wenn die tatsächliche Anzahl gesendeter Bytes (also die Puffergröße) s_{buffer} durch die Periodendauer t_{period} dividiert wird:

effektive Datenrate $r_{effective} = \frac{s_{buffer}}{t_{period}} = \frac{s_{buffer}}{t_{clear} + t_{fill}} = \frac{s_{buffer}}{\frac{s_{buffer} - l_{watermark}}{r_{controller}} + t_{fill}}$

a) Größe des Puffers: 1 Byte. Low-Watermark: 0 Byte.

$$r_{effective\ a} = \frac{s_{buffer\ a}}{\frac{s_{buffer\ a} - l_{watermark\ a}}{r_{controller}} + t_{fill}} = \frac{1\ \text{Byte}}{\frac{1\ \text{Byte} - 0\ \text{Byte}}{10000 \frac{\text{Byte}}{\text{s}}} + 5\ \text{ms}} \approx 196,078 \frac{\text{Byte}}{\text{s}} \approx 196 \frac{\text{Byte}}{\text{s}}$$

b) Größe des Puffers: 1 KiB = 1024 Byte. Low-Watermark: 0 Byte.

$$r_{effective\ b} = \frac{s_{buffer\ b}}{\frac{s_{buffer\ b} - l_{watermark\ b}}{r_{controller}} + t_{fill}} = \frac{1024\ \text{Byte}}{\frac{1024\ \text{Byte} - 0\ \text{Byte}}{10000 \frac{\text{Byte}}{\text{s}}} + 5\ \text{ms}} \approx 9534,451 \frac{\text{Byte}}{\text{s}} \approx 9534 \frac{\text{Byte}}{\text{s}}$$

c) Größe des Puffers: 1 KiB = 1024 Byte. Low-Watermark: 25 Byte.

$$r_{effective\ c} = \frac{s_{buffer\ c}}{\frac{s_{buffer\ c} - l_{watermark\ c}}{r_{controller}} + t_{fill}} = \frac{1024\ \text{Byte}}{\frac{1024\ \text{Byte} - 25\ \text{Byte}}{10000 \frac{\text{Byte}}{\text{s}}} + 5\ \text{ms}} \approx 9761,678 \frac{\text{Byte}}{\text{s}} \approx 9762 \frac{\text{Byte}}{\text{s}}$$

d)

> Wie müsste die Low-Watermark gewählt werden, damit die Netto-Datenrate des Geräts rechnerisch erreicht wird?

Es gilt also $r_{effective} = r_{controller}$. Die Formel für die effektive Datenrate muss nur zur **Low-Watermark** $l_{watermark}$ umgestellt werden:

$$r_{effective} = \frac{s_{buffer}}{\frac{s_{buffer} - l_{watermark}}{r_{controller}} + t_{fill}} \quad | \cdot \left(\frac{s_{buffer} - l_{watermark}}{r_{controller}} + t_{fill} \right) \tag{1}$$

$$r_{effective} \cdot \frac{s_{buffer} - l_{watermark}}{r_{controller}} + r_{effective} \cdot t_{fill} = s_{buffer} \quad | \cdot r_{controller} \tag{2}$$

$$r_{effective} \cdot s_{buffer} - r_{effective} \cdot l_{watermark} + r_{effective} \cdot t_{fill} \cdot r_{controller} = s_{buffer} \cdot r_{controller}$$
$$| + r_{effective} \cdot l_{watermark} - s_{buffer} \cdot r_{controller} \tag{3}$$

$$r_{effective} \cdot l_{watermark} = r_{effective} \cdot s_{buffer} + r_{effective} \cdot t_{fill} \cdot r_{controller} - s_{buffer} \cdot r_{controller}$$
$$| \div r_{effective} \tag{4}$$

$$l_{watermark} = s_{buffer} + t_{fill} \cdot r_{controller} - \frac{s_{buffer} \cdot r_{controller}}{r_{effective}} \quad |() \tag{5}$$

$$l_{watermark} = s_{buffer} \cdot \left(1 - \frac{r_{controller}}{r_{effective}} \right) + t_{fill} \cdot r_{controller} \tag{6}$$

Es ist ersichtlich, dass die **Low-Watermark** $l_{watermark}$ nur noch abhängig von der **Auffüllzeit des Puffers** t_{fill} und der **Datenrate des Gerätecontroller** $r_{controller}$ ist, da sich die Klammer zu 0 auflöst. Es ergibt sich daher folgender Wert:

$$l_{watermark} = 0 + t_{fill} \cdot r_{controller} = 5 \text{ ms} \cdot 10000 \frac{\text{Byte}}{\text{s}} = 50 \text{ Byte}$$

Allerdings muss der Puffer logischerweise Größer sein als die **Low-Watermark**. Dies ist nur für Aufgabenteile **b)** und **c)** der Fall.