Technische Informatik 2 WS 2016/17 C08
Tutor: Marc Hildebrandt

Timo Jasper (Inf, 3.FS.)
Thomas Tannous (Inf, 3.FS.)
Moritz Gerken (Inf, 3.FS.)

Übungsblatt 6

1	2	3	4	5	$\sum$

Lösungsvorschlag
Abgabe: 12.12.2016

Aufgabe 1

Unser Programm mycp funktioniert vollständig. Zunächst mussten wir die Datei auf Größe der zu kopierenden Datei vergrößern, damit das Mapping einwandfrei funktioniert. Die größe kriegen wir mit stat raus und die kopierte Datei haben wir mit ftruncate auf die entsprechende Größe gebracht. Um nun Block für Block zu kopieren brauchen wir eine Schleife. Davor müssen wir aber noch wissen wir groß ein Block sein muss. Der Block muss so groß wie eine Page sein, so haben wir die sysconf mit dem entsprechenden Parameter benutzt um die Pagesize zu kriegen. Nun gehen wir mit der While-Schleife Block für Block durch und mappen dementsprechend mit mmap die in und out Datei für den Addressbereich der sich von staddr bis staddr+pagesize erstreckt. Eine Ausnahme trifft auf, wenn die Startaddresse schon soweit hinten im Addressraum ist, dass eine Pagesize, die Größe der Datei überschreitet. So muss man für diesen fall noch die Startaddresse von der Gesamtgröße abziehen um die Größe zu kriegen, die genau die Datei abdeckt. Mit memcpy kopieren wir die bytes aus der source Datei die in den Arbeitsspeicher gemapped sind, dann zur Zieldatei. Anschließend benutzen wir munmap, um das Mapping zum Speicher wieder aufzulösen.

```
#include <string.h>
#include <iostream>
#include <cerrno>
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mman.h>

int main(int argc, char **argv){
    int fd_in, fd_out;
    struct stat *st;
    st = new struct stat;
    int filesize;
    long pagesize = sysconf(_SC_PAGESIZE);

    printf("%lu\n", pagesize);

    if (argc != 3) {
      std::cerr << "Usage: " << argv[0] << " source destination" <<
          std::endl;
      return -1;
    }
```

```
if ((fd_in = open(argv[1],O_RDONLY)) == -1){
  perror("open");
  return -1;
}
if(stat(argv[1], st) < 0) {
    close(fd_in);
    perror("couldnt stat file");
    return -1;
}

filesize = st->st_size;
//debug      printf("filesize = %i\n", filesize);

if ((fd_out = open(argv[2],O_RDWR|O_CREAT|O_TRUNC,S_IRUSR|S_IWUSR
   )) == -1){
  close(fd_in);
  perror("open");
  return -1;
}



if(ftruncate(fd_out,filesize) == -1) {
    close(fd_in);
    perror("couldnt truncate dest file");
    return -1;
}

int staddr = 0;
while(staddr < filesize) {



    if(filesize - staddr < pagesize) {
        pagesize = filesize - staddr;
    }

    //debug printf("page starts this time at: %i\n", staddr);

    char* inm = (char*) mmap(NULL, pagesize, PROT_READ,
       MAP_SHARED, fd_in, staddr);
    if (inm == MAP_FAILED) {
        close(fd_in);
        close(fd_out);
        printf("inmap failed map %s\n", strerror(errno));
        return -1;
    }
```

```
        char* outm = (char*) mmap(NULL, pagesize, PROT_WRITE |
           PROT_READ, MAP_SHARED, fd_out, staddr);
        if (outm == MAP_FAILED) {
            munmap(inm, pagesize);
            close(fd_in);
            close(fd_out);
            printf("outmap failed map %s\n", strerror(errno));
            return -1;
        }

        memcpy(outm, inm, pagesize);

        if (munmap(inm, pagesize) == -1) {
            close(fd_in);
            close(fd_out);
            printf("failed inmap unmap %s\n", strerror(errno));
            return -1;
        }

        if (munmap(outm, pagesize) == -1) {
            close(fd_in);
            close(fd_out);
            printf("failed outmap unmap %s\n", strerror(errno));
            return -1;
        }

        staddr += pagesize;
    }


    fsync(fd_out);
    close(fd_in);
    close(fd_out);

    return 0;
}
```

Aufgabe 2

a)

Im besten Fall haben alle Byte der Datei in Sektoren entsprechend ihrer Auslesereihenfolge auf einer Spur einer Platte zu liegen und der Lesekopf direkt davor. Somit wird nach dem auffinden des ersten Sektors keine zusätzliche Zeit benötigt um den nächsten auszulesenden Sektor zu finden, da er direkt der nächste ist. Sollten mehr Sektoren benötigt werden um die datei zu Speichern als auf einer Spur enthalten sind, müssen nach und nach die selben Spuren der anderen Platten belegt sein, da Kopfumschaltdauer kleiner Spurwechseldauer.Beim Umschalten von Kopf oder Spur gehen wir als best-case davon aus, dass die Sektoren so gelegt sind, dass der

Kopf genau vor der ersten Sektoren Position startet und damit 0 Zeit vergeht bis der richtige Sektor gefunden ist. Da es um den best-case geht gehen wir zusätzlich davon aus, dass zu beginn des Auslesens, die korrekte Spur gewählt ist und der Korrekte Kopf gewält ist und somit zu Beginn 0 Zeit vergeht das bis lesen auf der ersten korrekten Spur, am ersten Sektor beginnt.

$139586400B/512B = 272629,6875 Plattenbloecke$
wie viel Plattenblöcke sind betroffen ?
Dateigröße / Sektorgröße
Runden ist erst beim Endergebnisvorgesehn, obwohl im Massenspeicher nur ganze Blöcke belegt werden.

Lesedauer für eine komplette Spur auf einer Platte:
$(1/7200) * 60 * 1000 = 8,33333333ms$

Lesedauer für einen Sektor:
$(1/7200) * 60 * 1000/1200 = 0,006944444ms$

Lesedauer für alle Sektoren:
$0,006944444ms * 272629,6875 Sektoren = 1893,261597581ms$

Häufigkeit Lesekopfwechsel:
$(((272629,6875/1200)/8) * 7) = 198,792480469 \approx 199$ \\ muss logischerweise bereits jetzt aufgerundet werden, da auch für unvollständig belegte Spuren der entsprechende leseKopf komplett gewechselt sein muss.

Wechseldauer für alle Lesekopfwechsel:
$2,4ms * 199 wechsel = 477,6ms$

Häufigkeit Spurwechsel:
$(272629,6875/(1200 * 8)) = 28,398925781 \approx 29$ \\ muss logischerweise bereits jetzt aufgerundet werden, da auch für unvollständig belegte Spuren der leseKopf komplett auf der Spur befindlich sein muss.

Wechseldauer für alle Spurwechsel:
$4ms * 29 wechsel = 116ms$

Gesammtdauer:
$1893,261597581ms + 477,6ms + 116ms = 2486,861597581ms = \underline{2,486861598s}$

durchschnittliche Lesegeschwindigkeit(Datenrate):
$139586400B/2,486861598s = 56129540,989437885B/s$
$56129540,989437885B/s/1024B = 54814,004872498KiB/s = 53,529301633MiB/s$
$\underline{\approx 54814KiB/s \approx 54MiB/s}$

b)

beim worst-case befindet sich jeweils der nächste auszulesende Block niemals auf der Spur einer der Leseköpfe, es muss daher für jeden einzelnen Block ein spurwechsel stattfinden. Außerdem sind die Sektoren auf den spuren genau so verteilt, dass der Lesekopf immer direkt Hinter dem Start des gesuchten Sektors steht wenn er aktiviert wird und den großteil der Platte erst vorbeifahren lassen muss bevor er den Sektor von vorne lesen kann.

Blockwechsel:
worst-case(Blockwechsel*(Spurwechseldauer+rotationsdauer+Lesedauer))
$272630 * (4ms + 8,33331977ms + 0,006944444ms) = 3364326,23266282ms$

durchschnittliche Lesegeschwindigkeit(Datenrate):
$139586400B/3364326,23266282ms = 41,490149987B/ms$
$41,490149987B/ms * 1000ms/1024B = 40,517724597KiB/s = 0,03956809MiB/s$
$\approx 41KiB/s \approx 0.04MiB/s$

c)

Wir gehen davon aus, dass die logischen Blöcke weitgehend ausgefüllt sind und nicht nur als vergrößerung der bisherigen blöcke zu 7/8 leer sind.
Wenn immer 8 Blöcke zusammengefasst werden wird die Formel aus b) nur 1/8 mal so oft ausgeführt und sieht damit wie folgt aus :

Lesedauer für einen Logischen Block:
$(1/7200) * 60 * 1000/(1200/8) = 0,055555556ms$

Logischer Blockwechsel:
worst-case(Blockwechsel*(Spurwechseldauer+rotationsdauer+Lesedauer))
$\lceil(272630/8)\rceil * (4ms + 8,33331977ms + 0,055555556ms) = 422200,482234754$

durchschnittliche Lesegeschwindigkeit(Datenrate):
$139586400B/422200,482234754ms = 330,616391675B/ms$
$330,616391675B/ms * 1000/1024 =$
$322,867569995KiB/s = 0,315300361MiB/s$
$\approx 323KiB/s \approx 0,3MiB/s$

Aufgabe 3

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

Wenn Puffer leer ist wird ein Interrupt ausgelößt und der Puffer nachgefüllt, dies dauert 5ms. Der Puffer kann 1Byte an Daten aufnehmen.

Nach 5ms stehen dem Controller also wieder 1Byte an Daten zur verfügung.
Der Controller kann das Gerät mit einer Datenrate von 10000 Byte/s = 10 Byte/ms versorgen.
Der Controller leert den Puffer also in 0,1 ms.
Anschließend muss er wieder 5ms warten um wieder Daten im Puffer zu haben.
Es werden also 1Byte in 5,1 ms an das Gerät gegeben.
$Datenrate = 1Byte/5,1ms = 0,196078431B/ms = 196,078431373B/s \approx 196B/s.$

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

Ein Auffüllen des Puffers auf 1024 Byte dauert 5ms.
Der Controller leer den Puffer in $1024B/(10B/ms) = 102,4ms$.
und muss anschließend wieder 5ms warten.
Es werden also 1024 Byte in 107,4ms an das Gerät übergeben.
$Datenrate = 1024Byte/107,4ms = 9,534450652B/ms = 9534,450651769B/s \approx 9534B/s.$

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

Ein Auffüllen des Puffers auf 1024 Byte dauert 5ms.
Der Controller leer den Puffer in $1024B/(10B/ms) = 102,4ms$.
Die Watermark wird erreicht nach $999B/(10B/ms) = 99,9ms$.
Wenn keine Daten mehr im Puffer befindlich sind muss der Controller noch $102,4ms - 99,9ms = 2,5ms$ auf die durch den Interrupt angeforderte Puffer füllung warten.
Es werden also $1024Byte in 2,5ms + 102,4ms$ an das Gerät übergeben.
$Datenrate = 1024Byte/104,9ms = 9,761677788B/ms = 9761,67778837B/s \approx 9762B/s.$

Das initiale Füllen des Puffers , auf welches der Controller die vollen 5ms warten muss wurden hier nicht berücksichtigt, da sie die Datenrate abhängig von der Laufzeit des Controllers beeinflussen.

d)

Die Datenrate soll 10000 Byte/s betragen.
Wir gehen davon aus, dass die Puffergröße immernoch 1KiB beträgt.
Der Puffer wird also immernoch in 102,4ms geleert.
Nach $102,4ms - 5ms = 97,4ms$ muss also der Interrupt ausgelöst werden.
Nach 97,4ms befinden sich noch $1024B - (10B/ms) * 97,4ms = 50Byte$ im Puffer.
Die Watermark muss also auf 50Byte gesetzt werden.