Technische Informatik 2
Tutor: Marc Hildebrandt

WS 2016/17

C/06

Niklas Koenen
Jan Klüver
Vincent Jankovic

Übungsblatt 5

1	2	3	4	5	$\sum$

Lösungsvorschlag
Abgabe: 05.12.2016

Aufgabe 1

a)

Mit dem gdb kann man über die Aufrufe *disassemble main* bzw. *disassemble factorial* die virtuellen Adressräume ermitteln. Diese gehen von 0x000000000040057d bis 0x00000000004005e5 (main) bzw. von 0x000000000040054c bis 0x000000000040057c (factorial).
Dies ergibt insgesamt den Bereich 0x000000000040054c - 0x00000000004005e5.

b)

Es wird zunächst schonmal auf die Adresse des Stackpointers selber zugegriffen, also 0x7fffffffdf60.
In den nächsten Zeilen wird noch auf folgende Adresse zugegriffen:
0x7fffffffdf42 (mov %rdi,-0x18(%rbp))
0x7fffffffdf52 ($0x1,-0x8(%rbp))

c)

Die virtuelle Adresse des übergebenen Strings ist 0x400410. Hierzu kommt noch die Länge des Strings hinzu. Diese beträgt nach unserer Zählung 18 Bytes (17 + Nullbyte). Somit wird auf den Adressraum zwischen 0x400410 und 0x400428 zugegriffen.

d)

Offset Page-Frame 0: 0x1000000
Page-Größe: 4 KiB = 0x1000 B

- Virtuelle Adresse: 0x40054c
  Page(0x40054c)=floor(0x40054c/0x1000)=0x400=1024
  PF(1024)=17363=0x43d3
  Offset(0x40054c)=0x40054c mod 0x1000=0x54c
  Physische Adresse: 0x43d3 * 0x1000 + 0x1000000 + 0x54c=0x53d354c
- Virtuelle Adresse: 0x4005e5
  Page(0x4005e5)=floor(0x4005e5/0x1000)=0x400=1024
  PF(1024)=17363=0x43d3
  Offset(0x4005e5)=0x4005e5 mod 0x1000=0x5e5
  Physische Adresse: 0x43d3 * 0x1000 + 0x1000000 + 0x5e5=0x53d35e5

Also physischer Adressbereich von a): 0x53d354c - 0x53d35e5

- Virtuelle Adresse: 0x7fffffffdf60
  Page(0x7fffffffdf60)=floor(0x7fffffffdf60/0x1000)=0x7fffffffd=34359738365
  PF(34359738365):=2001=0x7d1 (hier gesetzt, da Page-Fault)
  Offset(0x7fffffffdf60)=0x7fffffffdf60 mod 0x1000=0xf60
  Physische Adresse: 0x7d1 * 0x1000 + 0x1000000 + 0xf60=0x17d1f60
- Virtuelle Adresse: 0x7fffffffdf42
  Page(0x7fffffffdf42)=floor(0x7fffffffdf42/0x1000)=0x7fffffffd=34359738365
  PF(34359738365)=2001=0x7d1
  Offset(0x7fffffffdf42)=0x7fffffffdf42 mod 0x1000=0xf42
  Physische Adresse: 0x7d1 * 0x1000 + 0x1000000 + 0xf42=0x17d1f42
- Virtuelle Adresse: 0x7fffffffdf52
  Page(0x7fffffffdf52)=floor(0x7fffffffdf52/0x1000)=0x7fffffffd=34359738365
  PF(34359738365)=2001=0x7d1
  Offset(0x7fffffffdf52)=0x7fffffffdf52 mod 0x1000=0xf52
  Physische Adresse: 0x7d1 * 0x1000 + 0x1000000 + 0xf42=0x17d1f42
- Virtuelle Adresse: 0x400410
  Page(0x400410)=floor(0x400410/0x1000)=0x400=1024
  PF(1024)=17363=0x43d3
  Offset(0x400410)=0x400410 mod 0x1000=0x410
  Physische Adresse: 0x43d3 * 0x1000 + 0x1000000 + 0x410=0x53d3410
- Virtuelle Adresse: 0x400428
  Page(0x400428)=floor(0x400428/0x1000)=0x400=1024
  PF(1024)=17363=0x43d3
  Offset(0x400428)=0x400428 mod 0x1000=0x428
  Physische Adresse: 0x43d3 * 0x1000 + 0x1000000 + 0x428=0x53d3428

Also ist der physische Adressbereich von c): 0x53d3410 - 0x53d3428

Aufgabe 2

In dieser Aufgabe sollten wir berechnen, wie viele Blöcke man mindestens benötigt, um eine Datei der größe 33.696.325 B auf der Platte zu speichern. Dabei ist jeder Datenblock 512 B und die Inode 128 B groß. Außerdem hat die Inode zehn direkt Einträge, einen Indirektblock, einen Zweifach-Indirektblock und einen Dreifach-Indirektblock. Zusätylich ist der Pointer auf den Speicherblock bzw. auf den Indirektblock 4 B groß.
Da jeder Indirektblock 512 B groß ist, können $\frac{512B}{4B} = 128$ indirekte Einträge auf einem Indirektblock gespeichert werden. Außerdem braucht man mindestens $\frac{33.696.325B}{512B} = 65.813,13$ d.h. 65.814 Blöcke, um allein die Datei zu speichern.
Da $10 + 128 + 128^2 = 16.522 < 65.814$ gilt, sind alle Datenblöcke im Indirektblock und im Zweifach-Indirektblock belegt. Also wird der Drefach-Indirektblock ebenfalls benötigt.
Es müessen somit $65.814 - 16.522 = 49.292$ Blöcke noch belegt werden. Nun ist die maximale Anzahl der Dreifach-Indirektblöcke 128^3, was deutlich zu viele Blöcke sind. Also gehen wir erstmal von der minimalen Anzahl aus, d.h jeder Dreifach-Indirektblock ist ein normaler Datenblock. Nun tauschen wir x dieser Datenblöcke gegen einen Indirektblock aus, der dann jeweils

128 neue Datenblöcke liefert. Damit gilt:

$$\begin{aligned} & 128^2 - x + 128x = 49.292 \\ \Leftrightarrow\ & x * (-1 + 128) = 49.292 - 128^2 \\ \Leftrightarrow\ & x = \frac{49.292 - 128^2}{127} = 259,118 \end{aligned}$$

Das bedeutet, es gibt 260 Dreifach-Indirektblöcke.
Insgesamt folgt also, dass es insgesamt $3 + 2 \cdot 128 + 260 = 519$ indirekte Blöcke und 65.814 Speicherblöcke geben muss. Da die Inode auch Speicherplatz verbraucht, werden insgesamt mindestens $65.814 + 519 + 1 = 66.334$ Speicherblöcke auf der Platte der Größe 512 B benötigt, um die ganze Datei speichern zu können.

Aufgabe 3

Zuerst werden mit dem Systemaufruf *open(...)* jeweils die gegebenen Dateien (Pfade) geöffnet und in g bzw. f gspeichert, wobei wir annhemen können, dass in beiden Fällen kein Fehler auftritt (es wird also keine -1 zurück gegeben). Der Dateideskriptor wird nun jeweils auf den entsprechenden Byte gesetzt, damit später gelesen werden kann. Dem Pfad der *nikolaus.avi* hat als 'access' Parameter read-only und die *meta* read and write. Die Inode von / wurde vom Bootsystem bereits ermittlet und wird als erstes, wenn nicht schon im Hauptspeicher (dies sollte eigentlich der Fall sein, beim booten), geladen. Danach wird die Inode von *home* geladen. Zusammen wurden also bisher Block 2 und Block 9 vom externen Speichermedium in den Hauptspeicher geladen. Ebenso werden die Inodes von *ti2, archive und nikolaus.avi* geladen (die Datenblöcke erst später) und der Deskriptor gesetzt. Diese besitzen neben den Metadaten der Dateien auch verweise auf die Datenblöcke, die evtl. wieder auf die nächste Inode in der Inodentabelle verweisen. Der Systemaufruf wird u"ber die Deskriptortabelle 'weitergeleitet', nähmlich an eine Filetabelle, die wiederum auf die Inodetabelle verweist bzw. auf die entsprechende Indode. Diese sollten also auch geladen werden. Dies geschiet auch (für jeden Prozess) fu"r die Systemaufrufe *lseek(...), read() und write()*. Sollten die Datenblöcke für diese Systemaufrufe noch nicht geladen worden sein, so passiert dies jetzt. Für *lseek()* wird Block Nr. 50 geladen, wobei dies auch ein anderer Datenblock sein kann. Wichtig ist, dass der Datenblock, der die letzten Bytes minus dem Offset von -10000 der Datei enthält (Pos+Offset= $fdPos \Leftrightarrow 121048576 - 10000 = 121038576$), geladen wird (evtl. werden auch mehr Datenblöke geladen, um das Ende zu ermitteln). Dies wird von *SEEKEND* impliziert. Den Block kann man analog zu Aufgabe 2 ermitteln. Die Nr. können wir aber nicht angeben. Hier wird nun der Filedeskriptor gesetzt. Mit *read* werden nun 4096 Bytes gelesen. Dazu müssen $\frac{4096}{512} = 8$ Blöcke geladen werden. Je nach dem wie die Datenblöcke sortiert sind, können es aber auch nur 7 sein. Die Daten der Datenblöcke werden nun in *count* gespeichert. Mit *write* wird nun der Inhalt von *count* in die von g induzierten Blöcke geschrieben. Um den Buffer müssen wir uns in dieser Aufgabe nicht kümmern, der Filedescriptor ist durch g gegeben. Da die Meta leer ist, werden zwar 7-8 'Meta' - Blöcke geladen und beschrieben, aber der erste Block in *meta* hat die Nr. 8521.