Technische Informatik 2
Tutor: Marc/Bingbin

WS 2016/17

C05

Tabea Eggers
Jan Fiedler
Florian Pflüger
Jonas Schmutte

Übungsblatt 5

Lösungsvorschlag
Abgabe: 05.12.2016

1	2	3	4	5	$\sum$
	2	2.25			

Aufgabe 1

a)

Der Adressbereich von *main*:

```
0x000000000040057d <+0>:
0x000000000040057e <+1>:
0x0000000000400581 <+4>:
0x0000000000400585 <+8>:
0x0000000000400588 <+11>:
0x000000000040058c <+15>:
0x0000000000400590 <+19>:
0x0000000000400592 <+21>:
0x0000000000400596 <+25>:
0x000000000040059a <+29>:
0x000000000040059d <+32>:
0x00000000004005a2 <+37>:
0x00000000004005a7 <+42>:
0x00000000004005aa <+45>:
0x00000000004005af <+50>:
0x00000000004005b1 <+52>:
0x00000000004005b6 <+57>:
0x00000000004005ba <+61>:
0x00000000004005be <+65>:
0x00000000004005c1 <+68>:
0x00000000004005c6 <+73>:
0x00000000004005c9 <+76>:
0x00000000004005cd <+80>:
0x00000000004005d0 <+83>:
0x00000000004005d5 <+88>:
0x00000000004005da <+93>:
0x00000000004005df <+98>:
0x00000000004005e4 <+103>:
0x00000000004005e5 <+104>:
```

Der Adressbereich von *factorial*:

```
0x000000000040054c <+0>:
0x000000000040054d <+1>:
0x0000000000400550 <+4>:
0x0000000000400554 <+8>:
0x000000000040055c <+16>:
0x000000000040055e <+18>:
0x0000000000400562 <+22>:
0x0000000000400567 <+27>:
0x000000000040056b <+31>:
0x0000000000400570 <+36>:
0x0000000000400575 <+41>:
0x0000000000400577 <+43>:
0x000000000040057b <+47>:
0x000000000040057c <+48>:
```

Dabei ist zu beachten, dass sich eine Adresse x, deren Nachfolgeradresse y nicht x+1 entspricht, über die Größe von y-x Adressen ausdehnt.

b)

Im Stacksegment wird auf die Adressen 0x7fffffffdf50 und 0x7fffffffdf40 zugegriffen.

Unser Lösungsweg:
Schritt für Schritt durch *factorial* debuggen und regelmäßig schauen, was in den Registern steht.
Bei Eintritt steht 0x7fffffffe668 in *%rsp*. Kurz vor dem mv-Befehl steht 0x7fffffffe660 in *%rsp*.
Es wurde also -0x8 gerechnet.
Dies wird dann in %rbp geschrieben.
Diese Werte verändern sich nun nicht mehr.
Es wird dann auf die Adressen -0x8(%rbp) und -0x18(%rbp) zugegriffen. Das Entspricht 0x7fffffffe658 und 0x7fffffffe648.

Bildet man diesen Vorgang mit 0x7fffffffdf60 als *%rsp* Startwert, kommt man auf Zugriffe auf die Adressen 0x7fffffffdf50 und 0x7fffffffdf40,

c)

Der von *printf* in *main* gelesene Format-String beginnt an 0x7fffffffe590. Diese Adresse wird in __printf errechnet und in %rsp gespeichert.

```
0x00007ffff7a9e190 <+0>:     sub    $0xd8,%rsp
0x00007ffff7a9e197 <+7>:     movzbl %al,%eax
0x00007ffff7a9e19a <+10>:    mov    %rdx,0x30(%rsp)
0x00007ffff7a9e19f <+15>:    lea    0x0(,%rax,4),%rdx
0x00007ffff7a9e1a7 <+23>:    lea    0x44(%rip),%rax
0x00007ffff7a9e1ae <+30>:    mov    %rsi,0x28(%rsp)
0x00007ffff7a9e1b3 <+35>:    mov    %rcx,0x38(%rsp)
0x00007ffff7a9e1b8 <+40>:    mov    %rdi,%rsi
0x00007ffff7a9e1bb <+43>:    sub    %rdx,%rax
0x00007ffff7a9e1be <+46>:    lea    0xcf(%rsp),%rdx
0x00007ffff7a9e1c6 <+54>:    mov    %r8,0x40(%rsp)
0x00007ffff7a9e1cb <+59>:    mov    %r9,0x48(%rsp)
```

Es wird dann noch im zusammenhang mit dieser Zeichenkette auf die Adressen 0x7fffffffe5c0 (0x30(%rsp)), 0x7fffffffe5b8 (0x28(%rsp)), 0x7fffffffe5c8 (0x38(%rsp)), 0x7fffffffe5d0 (0x40(%rsp)) und 0x7fffffffe5d8 (0x48(%rsp)) zugegriffen.

d)

Beispielrechnung für 0x40057d:
Pagegröße ist 4KiB also 4096B = 0x1000.
Die Page 0 startet beim Offset 0x1000000.
$Page : floor(0x40057d/0x1000) = 0x400 = 1024 \Rightarrow PF = 17363 = 0x43d3$
$Off : 0x40057d\%0x1000 = 0x57d$
$Phys.Adresse : 0x43d3 * 0x1000 + 0x57d + 0x1000000 = 0x53d357d$

Mit gleicher Rechnung ergibt sich für die Adressen aus *main*
$0x40057e \Rightarrow 0x53d357e$
$0x400581 \Rightarrow 0x53d3581$
$0x400585 \Rightarrow 0x53d3585$
$0x400588 \Rightarrow 0x53d3588$
$0x40058c \Rightarrow 0x53d358c$
$0x400590 \Rightarrow 0x53d3590$
$0x400592 \Rightarrow 0x53d3592$
$0x400596 \Rightarrow 0x53d3596$
$0x40059a \Rightarrow 0x53d359a$
$0x40059d \Rightarrow 0x53d359d$
$0x4005a2 \Rightarrow 0x53d35a2$
$0x4005a7 \Rightarrow 0x53d35a7$
$0x4005aa \Rightarrow 0x53d35aa$
$0x4005af \Rightarrow 0x53d35af$
$0x4005b1 \Rightarrow 0x53d35b1$
$0x4005b6 \Rightarrow 0x53d35b6$
$0x4005ba \Rightarrow 0x53d35ba$
$0x4005be \Rightarrow 0x53d35be$
$0x4005c1 \Rightarrow 0x53d35c1$
$0x4005c6 \Rightarrow 0x53d35c6$

$0x4005c9 \Rightarrow 0x53d35c9$
$0x4005cd \Rightarrow 0x53d35cd$
$0x4005d0 \Rightarrow 0x53d35d0$
$0x4005d5 \Rightarrow 0x53d35d5$
$0x4005da \Rightarrow 0x53d35da$
$0x4005df \Rightarrow 0x53d35df$
$0x4005e4 \Rightarrow 0x53d35e4$
$0x4005e5 \Rightarrow 0x53d35e5$

Für die aus *factorial*:
$0x40054c \Rightarrow 0x53d354c$
$0x40054d \Rightarrow 0x53d354d$
$0x400550 \Rightarrow 0x53d3550$
$0x400554 \Rightarrow 0x53d3554$
$0x40055c \Rightarrow 0x53d355c$
$0x40055e \Rightarrow 0x53d355e$
$0x400562 \Rightarrow 0x53d3562$
$0x400567 \Rightarrow 0x53d3567$
$0x40056b \Rightarrow 0x53d356b$
$0x400570 \Rightarrow 0x53d3570$
$0x400575 \Rightarrow 0x53d3575$
$0x400577 \Rightarrow 0x53d3577$
$0x40057b \Rightarrow 0x53d357b$
$0x40057c \Rightarrow 0x53d357c$

Für die Adressen aus b:
Hier haben wir pagenummer 4294967293 gewählt, da $Page(0x7fffffffdf50) = 0x7fffffffd = 34359738365$ nicht im Hauptspeicher ist.
Daher pageframe = 2001 = 0x7d1

$0x7fffffffdf50 \Rightarrow 0x7d1 * 0x1000 + f50 + 0x1000000 = 0x17d1f50$
$0x7fffffffdf40 \Rightarrow 0x17d1f40$

Für Adressen aus c:
Hier haben wir pagenummer 4195813 gewählt, da $Page(0x7fffffffe590) = 0x7fffffffe = 34359738366$ nicht im Hauptspeicher ist.
Daher pageframe = 100 = 64 $0x7fffffffe590 \Rightarrow 0x64 * 0x1000 + 0x590 + 0x1000000 = 0x1064590$
$0x7fffffffe5c0 \Rightarrow 0x10645c0$
$0x7fffffffe5b8 \Rightarrow 0x10645b8$
$0x7fffffffe5c8 \Rightarrow 0x10645c8$
$0x7fffffffe5d0 \Rightarrow 0x10645d0$
$0x7fffffffe5d8 \Rightarrow 0x10645d8$

Aufgabe 2 2/2P

Bei dieser Aufgabe haben wir mit der Gruppe `G05` zusammen gearbeitet.

Datei: 33696325 B
Datenblockgröße: 512 B
Berechnung der benötigten Datenblöcke: $33696325 : 512 = 65813,1347... \rightarrow$ 65814 Datenblöcke, da aufgerundet werden muss (sonst fehlen Daten)

Ein Indirektblock fasst 512 B.
Eine Datenblocknummer besteht aus 4 B.
Berechnung der möglichen Einträge in einem Indirektblock: $512 : 4 = 128$ Einträge

Eine Inode (= 1 Block) hat:
10 direkte Verweise
1 einfachen Indirektblock
1 doppelten Indirektblock
1 dreifachen Indirektblock

	Anzahl der Verweise auf Datenblöcke	benötigte Indirektblöcke
einfacher Indirektblock	128	1
doppelter Indirektblock	$128^2 = 16384$	$1 + 128$
dreifacher Indirektblock	$128^3 = 2097152$	$1 + 128 + 128^2$

einfacher + doppelter Indirektblock : 16502 mögliche Verweise
Berechnung der noch fehlenden Verweise: 65804 *(Da in der Inode schon 10 Direktverweise sind)* − 16502 = 49302 fehlende Einträge
Da die mögliche Anzahl von Verweisen beim dreifachen Indirektblock mehr als ausreichend ist, rechnet man:
$49302 : 128$ *(Anzahl der Einträge die ein Datenblock enthalten kann)* $= 385,1718... \rightarrow$ 386 noch benötigte Indirektblöcke der dritten "Ebene"(auch hier aufrunden)
um nun die zweite "Ebeneßu bekommen rechnet man: $386 : 128 = 3,0156... \rightarrow$ 4 Indirektblöcke in der zweiten "Ebene", wovon einer nicht vollständig ausgenutzt wird
Man erhält also für den dreifachen Indirektblock:

$$\underbrace{1}_{\substack{\text{Indirektblock}\\ \text{der ersten Ebene}}} + \underbrace{(3 + 3 \cdot 128)}_{\substack{\text{Indirektblock}\\ \text{der zweiten Ebene}}} + \underbrace{(1 + 2)}_{\substack{\text{Indirektblock der dritten Ebene mit Block aus zweiter Ebene der auf Block in dritter Ebene verweist}}} = 391 \text{ benötigte Indirektblöcke}$$

+1.5P

Nun lässt sich die Zahl der insgesamt benötigten Datenblöcke für die Datei berechnen:
65814 Datenblöcke (durch die Datei)
+ 1 Block (Inode)
+ 1 Indirektblock (einfacher Indirektblock)
+ 129 Indirektblöcke (doppelter Indirektblock)
+ 391 Indirektblöcke (dreifacher Indirektblock)

66336 Blöcke braucht man für die Datei insgesamt.

+0.5P

Aufgabe 3

Bei dieser Aufgabe haben wir mit der Gruppe F02 zusammen gearbeitet.

open("/home/ti2/archive/nikolaus.avi", O_RDONLY);

- Inode (0, /) ist in der Inode-Tabelle
- Datenblock von Inode 0 ist im Buffer-Cache
- Da Inode 0 ein Verzeichnis ist, steht da eine Tabelle mit den Dateinamen und den dazu gehörigen Inode-Nummern drin, welche in / sind. Diese Tabelle durchsuchen wir nach *home*.
- *home* hat die Inode-Nummer 36
- Inode 36, in die Inode-Tabelle eintragen und den dazugehörigen Datenblock 9 in den Cache laden.
- Da *home* ein Verzeichnis ist, haben wir also wieder eine Tabelle mit den nächsten Komponenten und deren Inode-Nummern. Suchen nach *ti2*
- *ti2* hat die Nummer 99.
- Also in die Inode-Tabelle eintragen und den Datenblock 2000 in den Cache laden
- *ti2* ist ein Verzeichnis welches wir dann nach den Eintrag *archive* durchsuchen. *archive* hat Inode-Nummer 206.
- Inode 206 eintragen und den Datenblock 3101 in den Cache holen

+1P

- Die Tabelle von *archive* nach *nikolaus.avi* durchsuchen, da *archive* ein Verzeichnis ist
- *nikolaus.avi* hat die Inode-Nummer 12783 , dann Einlesen von Block 50 für Inode 12783
- Inode in die Tabelle eintragen
- Nun kann der FileDiscriptor zurückgegeben werden, weil wir das Ziel erreicht haben (Es kann noch sein das die Datei schon geladen wird, auf den Verdacht hin das gleich auf diese zu gegriffen wird)

open("/home/ti2/meta", O_RDWR);

- Das gleiche Spiel wie eben durch die Inodes von / und *home* zu *ti2* navigieren, bloss befinden sich die Inodes schon in der Tabelle und die Datenblöcke im Cache.
- Den Datenblock von *ti2* durchsuchen wir nun nach *meta*

+1P

- *meta* hat die Inode-Nummer 112
- Die Inode tragen wir in die Tabelle ein
- Danach wird der FileDiscriptor zurückgegeben

lseek(f, -10000, SEEK_END);

- Schieb den FileDiscriptor an das 10000 vorletzte Byte von der *Nikolaus.avi*

read(f, buf, 4096);

- Liest 4096 Bytes aus den passenden Datenblöcke der *nikolaus.avi* und schreibt diese in *buf*
- und gibt die Anzahl der tatsächlich gelesenden Bytes zurück

write(g, buf, count);

- Nun werden die Bytes in *meta* geschrieben
- Danach werden die Metadaten in der Inode aktualisiert

+0.25P hinreichend korrekte Beschreibung der Auswirkungen von lseek() und read()/write() (insbesondere Zugriff auf Datenblöcke, die über Indirektblöcke erreicht werden)