BuS-Übungsgruppe 7 Übungsblatt 6 Felix Kiunke, 357322
Philipp Hochmann, 356148
Daniel Schleiz, 356092

5 Punkte **7.1**

a)

First-Fit: (512,256,3072) -> (389,256,3072) -> (0,256,3072) -> (0,256,2560) -> (0,256, 289)
Es werden die Anforderungen erfüllt.

Rotating-First-Fit: (512,256,3072) -> (389,256,3072) -> (389,256,2683) -> (389,256,2171)
Die letzte Anforderung von 2271 Bytes kann nicht erfüllt werden.

Best-Fit: (512,256,3072) -> (512,133,3072) -> (123,133,3072) -> (123,133,2560) -> (123,133,289)
Die Anforderungen werden erfüllt.

Worst-Fit: (512,256,3072) -> (512,256,2949) -> (512,256,2560) -> (512,256,2560) -> (512,256,2048)
Die letzte Anforderung von 2271 Bytes kann nicht erfüllt werden.

b)

- alle vier Strategien führen zum Erfolg:
ändere die Anforderungsreihenfolge: 2271 Byte, 512 Byte, 389 Byte, 123 Byte.
Speicherbereichsreihenfolge bleibt gleich: 512 Byte, 256 Byte, 3072 Byte.

First-Fit: (512,256,3072) -> (512,256,801) -> (0,256,801) -> (0,256,412) -> (0,133,412)
Rotating-First-Fit: (512,256,3072) -> (512,256,801) -> (0,256,801) -> (0,256,412) -> (0,133,412)
Best-Fit: (512,256,3072) -> (512,256,801) -> (0,256,801) -> (0,256,412) -> (0,133,412)
Worst-Fit: (512,256,3072) -> (512,256,801) -> (512,256,289) -> (123,256,289) -> (123,256,166)

-(First-Fit,Worst-Fit) und (Rotating-First-Fit,Best-Fit) sollen in jedem Schritt den selben Speicherbereich auswählen
ändere die Anforderungsreihenfolge: 2271 Byte, 512 Byte, 389 Byte, 123 Byte.
ändere Speicherbelegung: 3072 Byte als Erstes, dann 512 Byte, dann 256 Byte.

First-Fit: (3072,512,256) -> (801,512,256) -> (289,512,256) -> (289,123,256) -> (166,123,256)
Rotating-First-Fit: (3072,512,256) -> (801,512,256) -> (801,0,256) -> (412,0,256) -> (412,0,133)
Best-Fit: (3072,512,256) -> (801,512,256) -> (801,0,256) -> (412,0,256) -> (412,0,133)
Worst-Fit: (3072,512,256) -> (801,512,256) -> (289,512,256) -> (289,123,256) -> (166,123,256)

- nur Rotating-First-Fit und Best-Fit erfolgreich
ändere die Anforderungsreihenfolge: 512 Byte, 389 Byte, 2271 Byte, 123 Byte.
ändere Speicherbelegung: 3072 Byte als Erstes, dann 512 Byte, dann 256 Byte.
First-Fit: (3072,512,256) -> (2560,512,256) -> (2171,512,256) -> (2048,512,256)
Anforderung 2271 Byte kann nicht erfüllt werden und wird deshalb übersprungen, also nicht erfolgreich.

Rotating-First-Fit: (3072,512,256) -> (2560,512,256) -> (2560,123,256) -> (289,123,256) -> (289,0,256)
Alles erfüllt.

Best-Fit: (3072,512,256) -> (3072,0,256) -> (2683,0,256) -> (412,0,256) -> (412,0,133)
Alles erfüllt.

Worst-Fit: (3072,512,256) -> (2560,512,256) -> (2171,512,256) -> (2048,512,256)
Anforderung 2271 Byte kann nicht erfüllt werden und wird deshalb übersprungen, also nicht erfolgreich.

2,5 Punkte

7.2

a)
Entscheidend ist die Summe der Längen der jeweiligen Segmente;
Insgesamt stehen dem Prozess also 999+45+410+175+191+53+65 = 1938 Byte zur Verfügung.

b)
- physikalisch: 1895, logisch: (2,0).
kein Segmentation Fault, da Offset 0 < Länge 410 des Segments.

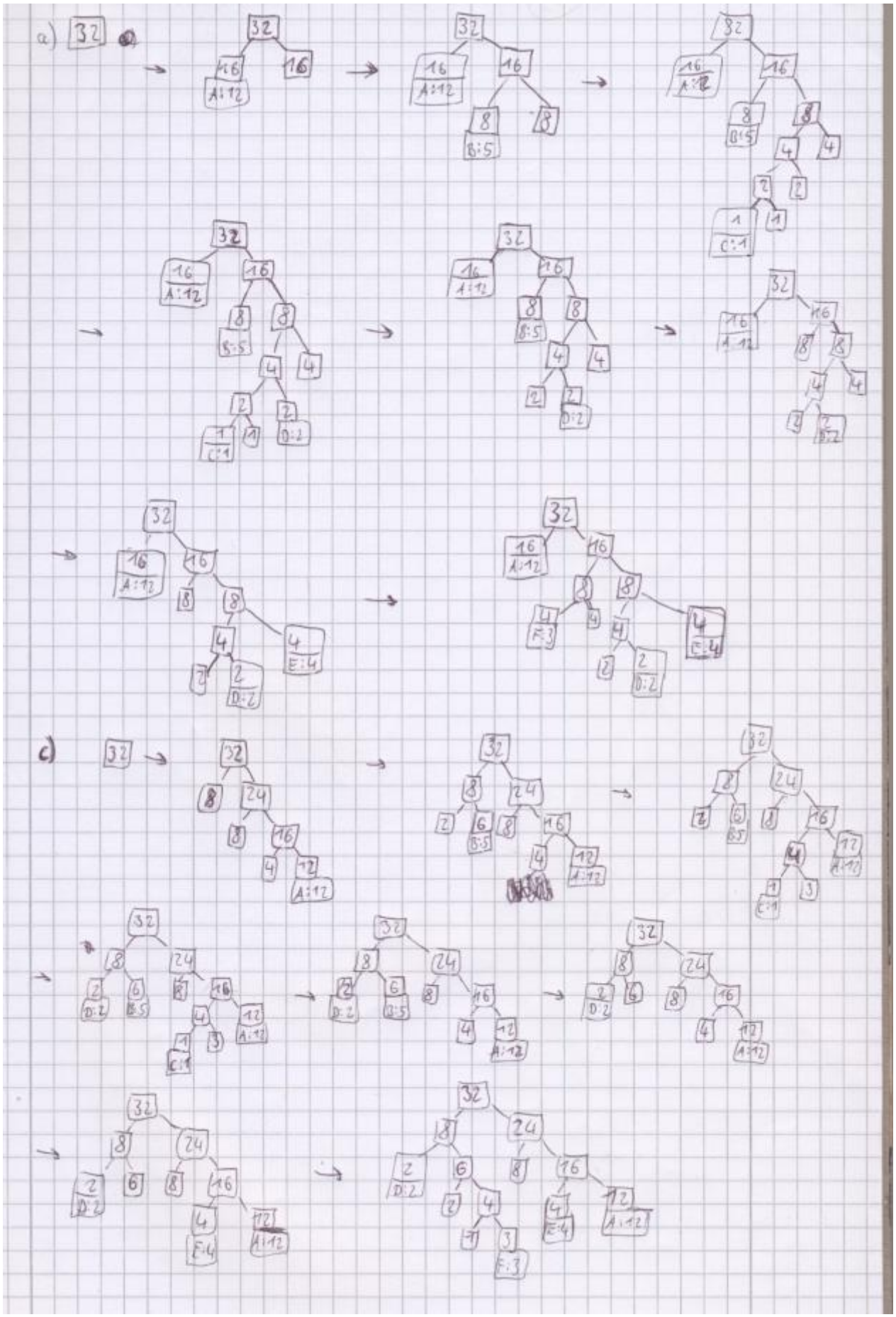
- physikalisch: 1575, logisch: (3,175).
Segmentation Fault, da Segment 3 nur Länge 175 hat und somit 1575 außerhalb liegt und sonst zu keinem anderen Segment gehört.

- physikalisch: 1935, logisch: (2,40)
kein Segmentation Fault, da Offset 40 < Länge 410 des Segments.

- physikalisch: 1050, logisch: (0,999).
Segmentation Fault, da Segment 0 nur Länge 999 hat und somit 1050 außerhalb liegt und sonst zu keinem anderen Segment gehört.

8 Punkte

7.3


a)
32
16
A:12
8
B:5
4
2
1
C:1
D:2
E:4
c)
24
12
A:12

b)
Die größtmögliche Speicheranforderung, die nun noch möglich ist, sind 4 MByte, da weder die 32, noch die 16, noch die 8 MByte Ebenen freien Platz haben.
Es gehen nach Ablauf aller Schritte 5 MByte Speicher durch interne Fragmentierung verloren, da einerseits A nur 12 von den zustehenden 16 MByte benutzt, und außerdem F nur 3 von den zur Verfügung stehenden 4 MByte.

d)
Die größtmögliche Speicheranforderung, die nun noch möglich ist, sind 8 MByte, da dies der größte freie Knoten ist.
Es geht nach Ablauf aller Schritte kein Speicher durch interne Fragmentierung verloren, da alle besetzten Knoten ihren Speicher komplett ausnutzen.

2,5 Punkte

7.4

a)
Es können 2^32 Bytes adressiert werden, ein KiB ist 1024 Byte groß. Man braucht also 2^32/(4*1024) = 1048576 pages.

b)
Eine Liste, in der jedes Element wieder genauso viele Elemente enthält:
x*x=1048576 <=> sqrt(1048576)=x=1024
Die Tabellen haben also Umfang 1024.

c)
Durch mehrstufiges Paging entstehen jeweils signifikant kleinere Page Tables, was letzten Endes dazu führt, dass die Suche effizienter ist, da eine Art Baumstruktur entsteht.
Als Nachteil wiederum entsteht aber für jede Stufe ein gewisser Overhead, also ein höherer Speicheraufwand zur Verwaltung.

2 Punkte

7.5

a)
Wenn eine Page verdrängt wird, in den bisher nur gelesen, und nicht geschrieben wurde, muss er nicht auf den Hintergrundspeicher geschrieben werden, weil er sich nicht geändert hat, und später einfach wieder aus der ursprünglichen Datenquelle geladen werden kann.

b)
Es werden nicht direkt alle Pages kopiert, sondern erst, wenn der Kindprozess versucht, den Inhalt einer Page zu verändern, es entsteht also weniger Aufwand.

c)
OPT kann nicht implementiert werden. OPT verdrängt immer die Seite, die am spätesten in der

Zukunft gebraucht wird. Das kann aber nicht verlässlich vorhergesagt werden, da die zukünftigen Zugriffe nicht fest bekannt sind.

d)
Sehr komplexe Algorithmen bringen auch oft den Nachteil mit sich, dass ein hoher Verwaltungsaufwand dabei entsteht:
Zum einen können unter Umständen die durch den Algorithmus verwendeten Datenstrukturen zu groß werden und generell ein Overhead entstehen lassen, zum anderen verbraucht ein komplexer Algorithmus auch dementsprechend viel Rechenzeit, die er den Prozessen zur Abarbeitung wegnimmt, es entsteht also eine Art Thrashing bei zu viel Verwaltungsaufwand.