 Mat.Nr.: 35 6507

1. Modul Informatik 3 · Winter 2016

18. November 2016

Wichtige Hinweise

- Schreiben Sie nicht in den Farben Rot oder Grün, und verwenden Sie keinen Bleistift.
- Schreiben Sie zunächst Ihre **Matrikelnummer** oben auf **jedes Blatt** dieser Klausur. Sie dürfen die Heftung auftrennen. Achten Sie auf gute Lesbarkeit!
- Beantworten Sie die Fragen auf dem dafür vorgesehenen Platz unter den Aufgaben. Sollte der Platz nicht ausreichen, fahren Sie **auf der Rückseite des Blattes** fort. Dort die Nummer der Aufgabe nicht vergessen!
- Es sind **keine Hilfsmittel** erlaubt, insbesondere kein eigener Block mit "Schmierpapier".
- Das Quiz dauert **30 Minuten**. Sie besteht aus 6 Aufgaben auf 7 Seiten. Es können maximal **29 Punkte** erreicht werden.

Viel Erfolg!

Aufgabe	Max.	Punkte
1	9	5
2	4	1
3	4	2
4	3	2.5
5	5	1,5
6	4	0
Gesamt	29	12

Note: _____

Modul Informatik 3
18. November 2016 Mat.Nr.: 55 6507

1. Aufgabe **9 Punkte**

1.1 Warum existieren in allen modernen Systemen Adress- und Datenbus nebeneinander? [1] 0

✓ 1.2 Erläutern Sie den Vorteil von DMA gegenüber Programmable-I/O. Beschreiben Sie hierfür kurz den Ablauf beim Laden von Daten einer Festplatte in den Hauptspeicher. [2] 2

✓ 1.3 Wollen DMA-Modul und CPU gleichzeitig das Bussystem nutzen, so wird stets das DMA-Modul bevorzugt. Warum? [1] 0

✓ 1.4 Erklären Sie in eigenen Worten kurz die Begrifflichkeiten Programm und Prozess, ihren Zusammenhang und ihre Unterschiede? [1] ?

✓ 1.5 Skizzieren Sie kurz den fünfstufigen Prozesslebenszyklus. Welche Status kommen bei der siebenstufigen Version hinzu und warum werden Sie benötigt? [4] 2

1.2 Die DMA ermöglicht es I/O direkt zw. Hauptspeicher und I/O-Controller über den Bus zu tätigen. Hierfür sendet die CPU eine Anfrage zum DMA Modul und kann sinnvoll weiterarbeiten bis die DMA bei Fertigstellen der angeordneten Aufgabe ein Interrupt sendet. Beim Programmierte I/O muss die CPU die ~~Aufgaben~~ Kommunikation zw. I/O-Controller und Hauptspeicher regeln, dieser aufwand entfällt bei DMA.

1.3 Weil I/O-Ops länger dauern und die CPU bzw. ein Prozess auf diese Daten wartet, sind weitere Prozesse an dieses wait geknüpft, so verstärkt sich der Effekt auf das System

1.4 Ein Programm ist eine Routine die für gleiche Eingaben, die gleiche Ausgabe liefert bei Korrektheit. Ein Prozess ist eine Instanz eines Programmes mit dem Entsp. Prozesskontrollblock

1.5 - new, Ready, Block, running, exit bei 5 Stufen

- new, ready, ready/Suspended, blocked, blocked/suspended, exit

Punkte

2. Aufgabe **4 Punkte**

Betrachten Sie das folgende Programm:

```
const int n=50;
int anzGesamt;

void total(){
        int count;
        for(count=0; count < n; count++) {
                if (count % 2 == 0)
                        anzGesamt++;
                else
                        anzGesamt--;
        }
}

int main(void) {
        anzGesamt=0;
        parbegin(total(), total());
        write(anzGesamt);
        return 0;
}
```

Anmerkung:
parbegin($P_1, P_2, \ldots, P_n$) bedeutet: Suspendiere die Ausführung des Main-Programms, starte die Prozeduren $P_1, P_2, \ldots, P_n$ nebenläufig und führe das Main-Programm weiter aus, nachdem alle
$P_1, P_2, \ldots, P_n$ terminiert sind.

2.1 Warum liegt hier eine Race Condition vor? [1] 1

2.2 Bestimmen Sie die korrekte untere und obere Grenze des Endwerts der Ausgabe der gemeinsamen Variablen *anzGesamt* durch dieses nebenläufige Programm. Begründen Sie ihre Ergebnisse. Nehmen Sie dafür an, dass Prozesse mit jeder beliebigen relativen Geschwindigkeit ausgeführt werden können und dass ein Wert nur verändert werden kann, nachdem er mit Hilfe eines separaten Maschinenbefehls in ein Register geladen wurde. [3] 0

Bonusmöglichkeit (2 Punkte):
Nehmen Sie an, dass unter Anwendung der Voraussetzungen in Teil (2.2) eine willkürliche Anzahl dieser Prozesse parallel ausgeführt werden darf. Welche Auswirkungen hat diese Änderung auf die Bandbreite der Endwerte von *anzGesamt*? Begründen Sie ihre Ergebnisse.

2.1 weil beide Instanzen von total() auf die gleichen Ressourcen zur gleichen Zeit zugreifen wollen. (✓)

3. Aufgabe **4 Punkte**

Das Lesen einer Datei von einem Dateiserver mit einem einzelnen Thread und einem Singlecore Multithread-Dateiserver soll verglichen werden. Es dauert 15 ms, um eine Arbeitsanfrage zu bekommen, sie weiterzuleiten und den Rest des notwendigen Ablaufs abzuarbeiten, wenn man annimmt, dass die benötigten Daten im Cache liegen. Wenn ein Festplattenzugriff notwendig ist, was in einem Drittel der Fälle vorkommt, so werden zusätzliche 75 ms benötigt. In dieser Zeit schläft der Thread.

✓ 3.1 Wie viele Anfragen kann der Server pro Sekunde bearbeiten, wenn er nur einen Thread hat? [2] 2

✓ 3.2 Wie viele Anfragen kann der Server pro Sekunde bearbeiten, wenn er mehrere Threads hat und Festplattenzugriffe parallel möglich sind? [2] 0

3.1 $T_{avg} = H\,T_{cache} + (1-H)(T_{mem} + T_{cache})$ $T_C = 15\,ms$ $T_{Mem} = 75\,ms$ $H = \frac{2}{3}$

$T_{avg} = \frac{2}{3} \cdot 15\,ms + \left(1 - \frac{2}{3}\right)(90\,ms)$

$= 10\,ms + 30\,ms = 40\,ms$

$\Rightarrow$ Pro Sekunde $\frac{1000\,ms}{40\,ms} = 25$ Anfragen/s ✓

3.2

(Anzahl Threads · (25 Anfragen/s)) − ~~Overhead~~ $\sum_{i=1}^{\text{Anz. Threads}}$ Overhead 6

4. Aufgabe **3 Punkte**

4.1 Was muss innerhalb des Betriebssystems vorbereitend getan werden, damit auf einen Interrupt reagiert werden kann? [1] 1

4.2 Skizzieren Sie kurz die Aktionen, die nach dem Auftreten eines Interrupts ablaufen, nach dem Eintreffen des Signals in der CPU. Nehmen Sie als Beispiel einen Tastendruck an der Tastatur. (Interruptsignalwert 1) [2] 1.5

4.1. • Ein entsprechender Treiber muss installiert sein
quasi. Der Handler
•

4.2 • Hardware Interrupt → Interrupt Handler → Ordnet Signal richtigem Handler zu → Treiber entspr. Interrupt
je nach Treiber, hier...
- Systemlib/ Betriebssystem Call

~~Unter Linux würde hier der Entspr. Signalwert~~
~~in~~

fehlt: Unterbrechung des laufenden Prozesses

5. Aufgabe **5 Punkte**

Beim folgenden Programm haben sich verschiedene Fehler eingeschlichen. Finden Sie diese (inklusive Compiler-Warnungen) und erklären Sie warum es ein Fehler ist und wie er beseitigt werden könnte. Als Ausgabe wird eine gültige ASCII-Zeichenkette erwartet. Wie könnte diese aussehen? Nutzen Sie hierzu auch den unteren Ausschnitt der ASCII-Tabelle.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void main(void) {
5          char *line = "Das ist eine Textzeile!";
6          char *p = line;
7          int n = sizeof(line);
8          int i;
9
10         for (i = 0; i <= n; i++) {
11                 if (line[i] < 91 && line[i] > 64)
12                         line[i] |= 128;
13
14         }
15
16         while (*p & 0x7F != '\0') {
17                 if (*p & 128)
18                         *p = '*';
19                 *p += 1;
20         }
21
22         printf("%s\n", line);
23
24         return 0;
25 }
```

1 // int main(...) f. Rückgabe v. Fehlern im OS

// größe des Pointers, nicht dessen Worauf er zeigt 0,5

Wenn line[i] Buchstabe

// Wenn der String per Pointer deklariert und initialisiert wurde, kann man nicht mit dieser Schreibweise zugreifen f

// Was ist 0x7F? Überhaupt def.? s Tabelle

// immer True

// *p='*' wäre eine Zuw. d. Werte; *p='*' setzt den Pointer auf '*' f

0x7 = 0111₂

Oct	Dec	Hex	Char	Oct	Dec	Hex	Char
000	0	00	NUL '\0' (null character)	100	64	40	@
001	1	01	SOH (start of heading)	101	65	41	A
002	2	02	STX (start of text)	102	66	42	B
003	3	03	ETX (end of text)	103	67	43	C
004	4	04	EOT (end of transmission)	104	68	44	D
...							
026	22	16	SYN (synchronous idle)	126	86	56	V
027	23	17	ETB (end of trans. blk)	127	87	57	W
030	24	18	CAN (cancel)	130	88	58	X
031	25	19	EM (end of medium)	131	89	59	Y
032	26	1A	SUB (substitute)	132	90	5A	Z

Ausgabe? Compiliert das Programm überhaupt?
Nachdem i.- alle Fehler korrigiert habt, ja

Punkte (1,5)

6. Aufgabe **4 Punkte**

Schreiben Sie ein Programm in Pseudocode dass im GPIO-Modul 1 des BeagleBones die erste LED (Pin 21) anschaltet. Die Anfangsadresse des zum Modul gehörenden Speicherbereichs ist `0x4804C000` und die Gesamtgröße `0x1000`. Das `GPIO_SETDATAOUT` Register des GPIO-Moduls hat den Offset `0x194` und funktioniert wie folgt: "Writing a 1 to a bit in the GPIO_SETDATAOUT register sets to 1 the corresponding bit in the GPIO_DATAOUT register; writing a 0 has no effect". Pin n wird dabei vom n-ten Bit gesteuert. Nutzen Sie `open` und `mmap`.

```
int main (void) { System("echo default-on > /sys/class/leds/beaglebone
                                                    ...:usr0/
                                                    trigger")
```

war nicht gefragt

Punkte ∅

 Mat.Nr.: 856507

1. Modul Informatik 3 · Winter 2016

16. Dezember 2016

Wichtige Hinweise

- Schreiben Sie nicht in den Farben Rot oder Grün, und verwenden Sie keinen Bleistift.
- Schreiben Sie zunächst Ihre **Matrikelnummer** oben auf **jedes Blatt** dieser Klausur. Sie dürfen die Heftung auftrennen. Achten Sie auf gute Lesbarkeit!
- Beantworten Sie die Fragen auf dem dafür vorgesehenen Platz unter den Aufgaben. Sollte der Platz nicht ausreichen, fahren Sie **auf der Rückseite des Blattes** fort. Dort die Nummer der Aufgabe nicht vergessen!
- Es sind **keine Hilfsmittel** erlaubt, insbesondere kein eigener Block mit "Schmierpapier".
- Das Quiz dauert **36 Minuten**. Sie besteht aus 5 Aufgaben auf 6 Seiten. Es können maximal **29 Punkte** erreicht werden.

Viel Erfolg!

Klaus: Überlappende Aufg
z.B. Race -> Mutex -> Deadlock

Aufgabe	Max.	Punkte
1	6	2
2	8	5
3	5	4.5
4	6	3,5
5	4	2
Gesamt	29	17

Note: _____

1. Aufgabe **6 Punkte**

✓ 1.1	Was versteht man unter synchronem Nachrichtentransfer im Sinne des Message-Passing?	1	1
✗ 1.2	Erklären Sie die Unterschiede von User-Level-Threads gegenüber Kernel-Threads.	2	0
✓ 1.3	Gibt es Geschwindigkeitsunterschiede zwischen einem Processswitch und einen Threadswitch, wenn letzterer zwischen zwei Threads des selben Prozesses geschieht? Begründen Sie.	1	1
✓ 1.4	Der Minix-Scheduler verwendet in jeder Prioritätsebene ein System, welches an Round Robin angelehnt ist, aber ein großes Problem dieses Algorithmus' beseitigt. Welches Problem ist gemeint und wodurch wird es ausgehebelt? (Was ist der Unterschied zwischen echtem Round Robin und einer Queueebene in Minix?)	2	0

1.1. Synchron Msg-passing: Sender und Empfänger versenden/empfangen die Antwort blocking, d.h. Der sender schickt die Nachricht bis sie als zugestellt best. wurde. Der Empfänger wartet bis er eine Nachricht empfängt und sendet dann eine Bestätigung back ✓

1.3. • Ja, denn beim Thread switch muss lediglich der kleinere Thread kontrollblock ausgetauscht werden. Beim Prozess switch muss der gesamte ~~Thread kont~~ Prozess kontrollblock neu geladen werden, der deutlich mehr Daten enthält.

1.4. Starvation durch kontinuierliches absinken/Eingabe von höher Priorisierten Prozessen. Lösung: ~~Aged~~ balance_queues: hebt die Priorität (eines Proc) automatisch alle 5 sekunden, wenn dieser nicht schon in seiner max Prio queue liegt

2.3 Circular Wait, Mutex, No preemption, Hold & Wait

2. Aufgabe **8 Punkte**

Betrachten Sie das folgende Programm:

2.1

```
const int n=50;
int anzGesamt;

void totall(){
        int count;
        for(count=0; count < n; count++) {
                anzGesamt++;
        }
}

void total2(){
        int count;
        for(count=0; count < n; count++) {
                anzGesamt--;
        }
}

int main(void) {
        anzGesamt=0;
        parbegin(totall(), total2());
        write(anzGesamt);
        return 0;
}
```

int lock=0;
SemWait(lock);
SemSig(lock);
SemWait(lock);
SemSig(lock);

2.3

int A=0, B=0;
wait(A); wait(B); } Hold and Wait Mutex No preemption je nach Sched.
Sig(A); Sig(B);
Circ Wait wenn total 1 vor dann total 2 bzw dann wartet total auf B und total 2 auf A
wait(B); wait(A);
Sig(B); Sig(A);

Anmerkung:
parbegin($P_1, P_2, \ldots, P_n$) bedeutet: Suspendiere die Ausführung des Main-Programms, starte die Prozeduren $P_1, P_2, \ldots, P_n$ nebenläufig und führe das Main-Programm weiter aus, nachdem alle
$P_1, P_2, \ldots, P_n$ terminiert sind.

✓ 2.1 Bei diesem Code liegt eine Race Condition vor. Beheben Sie diese durch die Nutzung eines binären Semaphors. [1] 1

✓ 2.2 Was Sind die minimal und maximal möglichen Ergebnisse von *anzGesamt* bei paralleler Ausführung vor ihrer Anpassung in Aufgabenteil 1 und wie verändern sich diese nach diesen? Begründen Sie jeweils. [3] 3

✓ 2.3 Konstruieren Sie eine Lösung, die anfällig für Deadlocks ist. Verwenden Sie hierfür zwei Semaphoren. Benennen Sie die Vorraussetzungen für einen Deadlock, die Sie dadurch hergestellt haben, weshalb Sie nun vorliegen und erläutern Sie diese kurz. [4] 7

2.2: -50 bis +50, denn ~~totall()~~ total2() kann ~~das~~ die Increment-V. total1 immer überschreiben ~~und~~ mit einer Dekrementierung und anders-rum. ~~Ein Fehler~~ Nach der Änderung tritt das Ergebnis 0 auf, denn sowohl lesen wie auch schreiben ~~kann~~ ist erst möglich, wenn die andere Funktion den kritischen Bereich wieder freigibt.

3.3: Durch I/O-Bound-Prozesse kommt es zu Wartezeiten, diese können nicht minimiert werden, denn sie sind z.T. nicht vorhersehbar (Trap => beheben => Wartezeit f. alle Proz.), nicht vermeidbar (2 Proz. greifen abwechselnd auf gleiche Ressource zu z.B.) Keine Zukunftsaussage

3. Aufgabe 5 Punkte

Gegeben seien die folgenden Prozesse, ihre Ankunfts- und Bearbeitungszeiten.

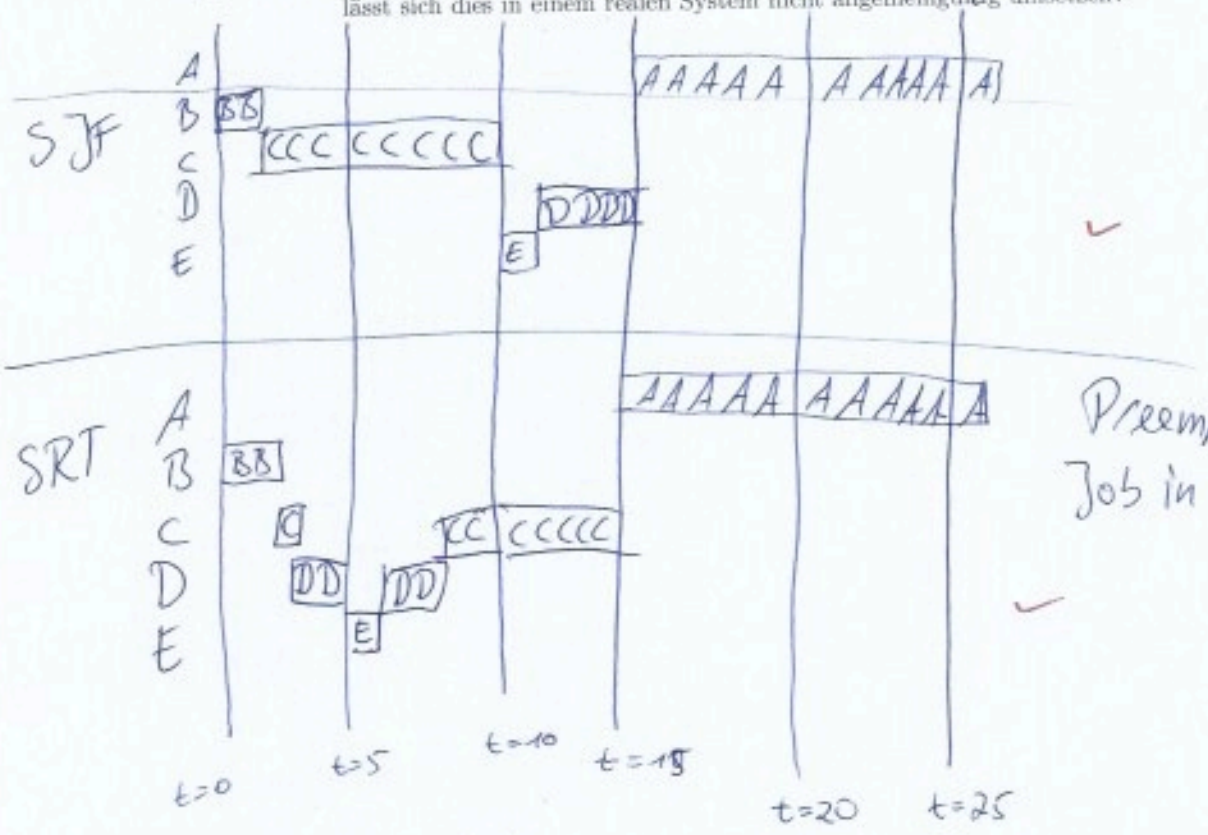
- A erscheint zum Zeitpunkt 0 und benötigt 11 Zeiteinheiten zum Beenden.
- B erscheint zum Zeitpunkt 0 und benötigt 2 Zeiteinheiten zum Beenden.
- C erscheint zum Zeitpunkt 1 und benötigt 8 Zeiteinheiten zum Beenden.
- D erscheint zum Zeitpunkt 3 und benötigt 4 Zeiteinheit zum Beenden.
- E erscheint zum Zeitpunkt 5 und benötigt eine Zeiteinheit.

Ein Prozess ist sofort zum Erscheinungszeitpunkt lauffähig, kann also vom Scheduler für die Ausführung in der CPU ausgewählt werden und sofort laufen, sollte es das Schedulingverfahren ermöglichen.

3.1 Skizzieren Sie die Ausführungsreihenfolge der Prozesse A bis E für die beiden Schedulingalgorithmen Shortest Job First (SJF) und Shortest Remaining Time (SRT). Zeigen Sie weiterhin, worin sich diese beiden Algorithmen unterscheiden. [3] 3

3.2 Sowohl SRT, als auch SJF, sind anfällig für ein Problem vieler Schedulingalgorithmen. Welches? Erläutern Sie kurz. [1] 1

3.3 Ein perfektes Schedulingverfahren würde die durchschnittliche Wartezeit über alle Prozesse minimieren, so dass es keine andere Ausführungsreihenfolge gibt, die für den Durchschnitt des Verhältnisses von Systemzeit zu CPUzeit einen geringeren Wert erzielen kann. Warum lässt sich dies in einem realen System nicht allgemeingültig umsetzen? [1] 0.5



Preemption wenn kürzerer Job in rdy übergeht

3.2 Starvation: Wenn immer kürzere Prozesse in den rdy-Zustand übergehen, so werden zunächst/als nächstes diese Ausgeführt => sehr langer Prozess wird nie CPU-ausgef.

Punkte

4. Aufgabe **6 Punkte**

Beantworten Sie folgende Fragen in der Annahme, dass der Code mit den gleichen Compiler-Flags kompiliert wird wie in der Vorlesung gezeigt:

1. Kompiliert der Code ohne Fehler? Wenn nicht, was müsste man ändern? ○ Compiliert
2. Was ist der Zweck der Funktionen `A`, `B` und `C`? 2
3. Was ist die erwartete Ausgabe des Programms? 1,5

```
1  #include <stdio.h>
2
3  const int c = 10;
4
5  int A(int a, int b) {
6      int c = a+b;
7
8      while(c > 10) {
9          int a = c-11;
10
11         c = a + 1;
12
13         printf("%d, %d, %d\n", a, b, c);
14     }
15
16     printf("a: %d\n", a);
17
18     return c;
19 }
20
21 int B(int c) {
22     return c+c;
23 }
24
25 int C(const char *d) {
26     return (int) *(d+c);
27 }
28
29 int main(void) {
30     int a = 17, b = 4, c = 3;
31     const char d[] = "Das ist ein toller String!";
32
33     printf("A: %d\n", A(a, b));
34     printf("B: %d\n", B(c));
35     printf("C: %d\n", C(d));
36
37     return 0;
38 }
```

Zweck?
21, ~~Zweimal rum, um 11 dek~~
c wird um 10 verringert pro DL bis c > 10, also a+b % 10 ✓

(mod) 1. DL: 17, 4, 11 | 0
2. DL: 17, 4, 1 | 0 F

a: 0 ✓

// Verdopple c ✓

// y c hier nicht def. F
// ~~gib die Summe des ersten~~ gib einen Pointer auf den c. Buchstaben von d aus, also d[c] in int-Ascii/encoding-Wert ✓

in int Darst. Ascii-Wert

A: 1 ✓
B: 96 ✓

Punkte
3,5

5. Aufgabe **4 Punkte**

Wie sieht die Ausgabe des nachfolgenden Programms aus? Schreiben Sie die Ausgabe der einzelnen `printf`-Aufrufe jeweils dahinter. Nehmen Sie an, dass das Programm lauffähig und ohne Warnungen kompiliert.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <err.h>
5
6  void f(int *a)
7  {
8
9          printf("%zu\n", sizeof(a));
10
11
12         printf("%zu\n", sizeof(*a));
13
14
15         printf("%zu\n", sizeof(a[0]));
16
17
18
19         int b[10][5] = {{1, 2, 3, 4, 5}, {6, 7, 8, 9}};
20
21         printf("%zu\n", sizeof(b));
22
23
24         printf("%zu\n", sizeof(*b));
25
26
27         printf("%zu\n", sizeof(&b));
28
29
30         printf("%zu\n", sizeof(*(b[0] + 1)));
31
32
33 }
34
35 int main(void)
36 {
37         /* Beispielausgabe */
38         printf("%zu\n", sizeof(int));       /* Ausgabe 4 */
39         printf("%zu\n", sizeof(void *));    /* Ausgabe 8 */
40
41         int a[5] = {1, 2, 3};
42
43         printf("%zu\n", sizeof(a));
44
45         f(a);
46
47         return 0;
48 }
```

/* Ausg 8 */ ✓ gr. Pointer

/* Ausg 4 */ ✓ gr Wort => Int = 4

/* Ausg 4 */ ✓ — " —

/* Ausg 200 */ F 200 10·5·4

/* Ausg 20 */ F 20 1. Inneres Arr. 5·4 = 20

/* Ausg 8 */ ✓ Adresse

/* Ausg 8 */ F 4 Integer; 2. Elem

/* Ausg 8 */ F 20 5·4 ints

Punkte 2