technische universität
dortmund

Betriebssysteme

Tafelübung 4. Speicherverwaltung

http://ess.cs.tu-dortmund.de/DE/Teaching/SS2017/BS/

Olaf Spinczyk

olaf.spinczyk@tu-dortmund.de
http://ess.cs.tu-dortmund.de/~os

AG Eingebettete Systemsoftware
Informatik 12, TU Dortmund

Agenda

- Besprechung Aufgabe 3 – Deadlocks
- Dynamische Speicherverwaltung in C
- Debugging mit gdb
- Buddy-Verfahren
- Aufgabe 4 – Speicherverwaltung
- Einfache Tests in C
- Klausuraufgabe

Besprechung Aufgabe 3

→ Foliensatz Besprechung

Dynamische Speicherverwaltung in C

- **malloc** („memory alloc“): Standardbibliotheksfunktion, reserviert dynamisch Speicher auf dem *Heap*

Dynamische Speicherverwaltung in C

- **malloc** („memory alloc“): Standardbibliotheksfunktion, reserviert dynamisch Speicher auf dem *Heap*
- aus **malloc(3)**:
  - **void** *malloc(size_t size)
  - reserviert *size* Bytes
  - liefert einen Pointer auf den Anfang des Speicherbereichs
  - oder im Fehlerfall (!): NULL
  - size_t: plattformunabhängiger Typ für Speicherbereichsgrößen (sizeof() ist z.B. auch vom Typ size_t!)

Dynamische Speicherverwaltung in C

- **malloc** („memory alloc“): Standardbibliotheksfunktion, reserviert dynamisch Speicher auf dem *Heap*
- aus **malloc(3)**:
  - **void** *malloc(size_t size)
  - reserviert *size* Bytes
  - liefert einen Pointer auf den Anfang des Speicherbereichs
  - oder im Fehlerfall (!): NULL
  - size_t: plattformunabhängiger Typ für Speicherbereichsgrößen (sizeof() ist z.B. auch vom Typ size_t!)
- **free**: gibt zuvor mit malloc (oder calloc/realloc) belegten Speicher wieder frei
  - **void** free(**void** *ptr)
  - Speicher darf nur 1x free()d werden!

Dynamische Speicherverwaltung in C

- Beispiel für malloc/free:

```
int *first_n_squares(unsigned n) {
    int *array, i;
    array = malloc(n * sizeof(int));    /* kein Cast notwendig! */
    if (array == NULL) {                /* Fehlerbehandlung */
        perror("malloc");
        exit(EXIT_FAILURE);
    }
    for (i = 0; i < n; ++i)          /* Array befüllen ... */
        array[i] = i * i;
    return array;                       /* ... und zurückliefern */
}   /* Die Variable "array" hoert hier auf zu existieren – nicht *
     * aber der Speicherbereich, auf den sie zeigt!              */

int main(void) {
    int *ptr;
    /* ... */
    ptr = first_n_squares(200);
    printf("10*10 = %d\n", ptr[10]);
    free(ptr);
    return 0;
}
```

Dynamische Speicherverwaltung in C

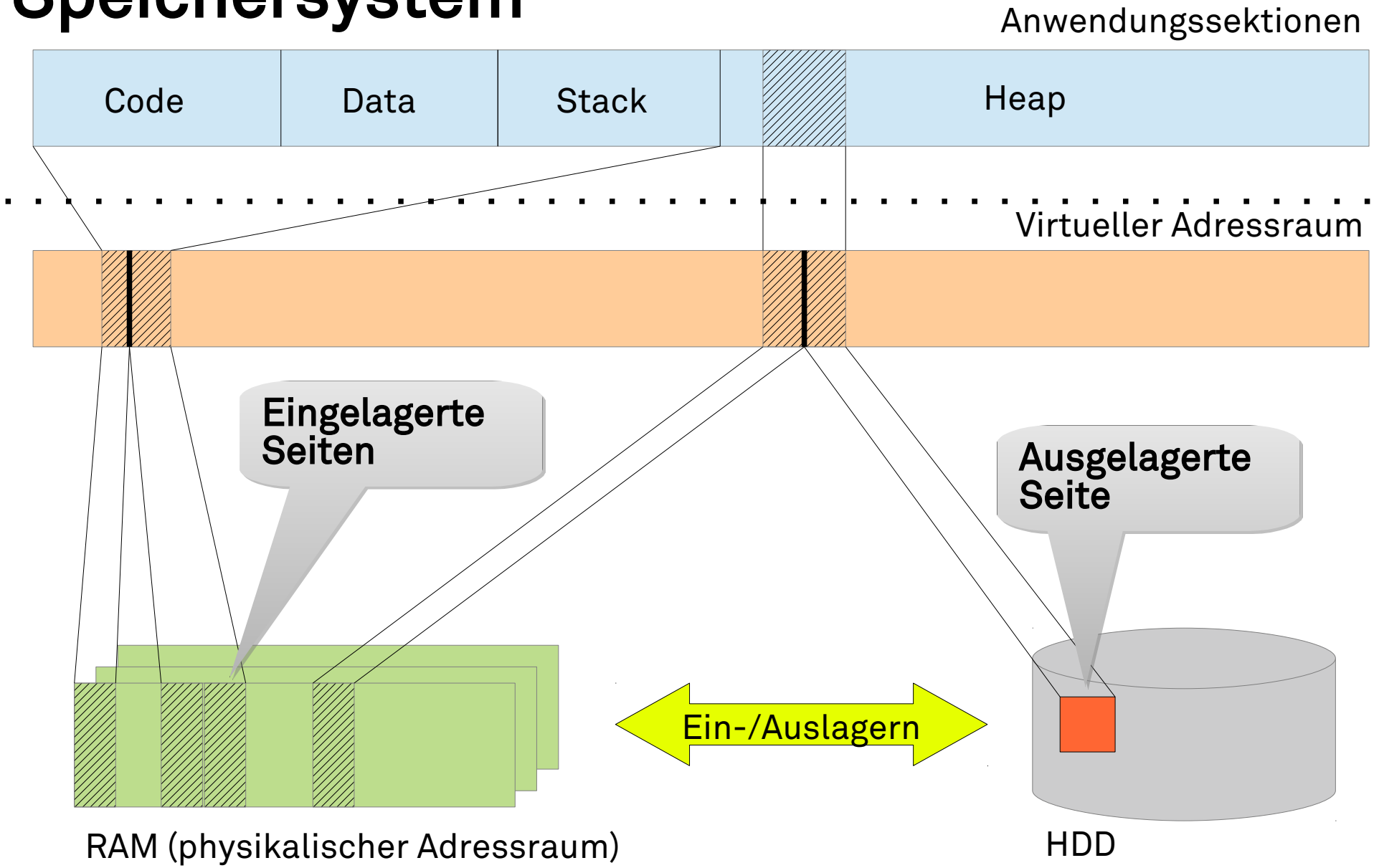
```
char *ptr = malloc(42);
```

- ptr zeigt jetzt auf einen Speicherbereich der Länge 42
- Wie schreiben wir ein **int** mit dem Wert 0x12345678 an den Anfang?
  - `ptr` vom Typ **`char*`** („Zeiger auf **char**“)
  - `(int *)ptr` Cast auf den Typ **`int*`** („Zeiger auf **int**“)
  - und ab da wie üblich: Dereferenzieren (***** davor), um den Wert „anfassen“ zu können, auf den der Zeiger zeigt! (Klammerung!)
  - `*((int*)ptr)` ist vom Typ **int** (**`char*`**, auf **`int*`** gecastet, dereferenziert)

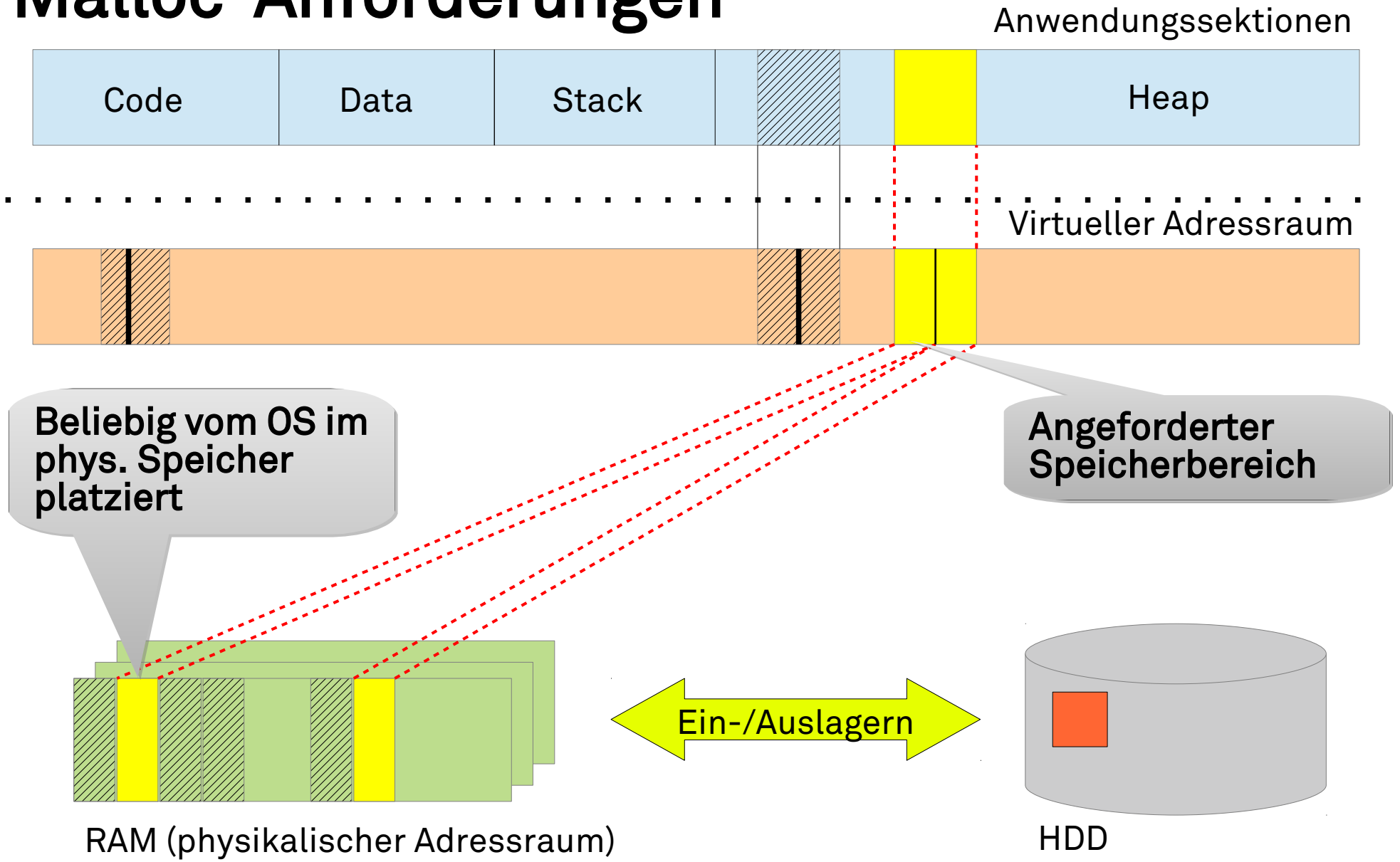
```
*((int*)ptr) = 0x12345678;
```

  - Quizfrage: In welchem Bereich des *Speicherlayouts* landet der angeforderte Speicher?

Speichersystem



Malloc-Anforderungen



Pointerarithmetik

- Noch eine abschließende Quizfrage zu Pointerarithmetik ...
- Was ist der Unterschied zwischen ...

```
/* ptr hat einen gueltigen Wert */
return ((char *)ptr) + 1;
```

- ... und ...

```
/* ptr hat einen gueltigen Wert */
return ((int *)ptr) + 1;
```

- ... ?

Pointerarithmetik

- Noch eine abschließende Quizfrage zu Pointerarithmetik ...
- Was ist der Unterschied zwischen ...

```
/* ptr hat einen gueltigen Wert */
return ((char *)ptr) + 1;
```

- ... und ...

```
/* ptr hat einen gueltigen Wert */
return ((int *)ptr) + 1;
```

- ... ?
- Antwort: Bei Berechnungen (Addition, Subtraktion, ...) mit Zeigern hängt die Adressdifferenz vom Zeigertyp ab!
  - Erhöhung um **sizeof**(**char**) vs. **sizeof**(**int**)!

Backtraces mit gdb

- **GDB** – The Gnu Project Debugger
- Nützlich zum Debuggen von Programmabstürzen
- Benötigt Debug-Informationen (GCC mit -g aufrufen)

Ausführbare Datei

Absturzposition im Code

```
derf@vatos A4$ gcc -Wall -g -o test_4a …
derf@vatos A4$ gdb ./test_4a
(gdb) run
Program received signal SIGSEGV, Segmentation fault.
0x0000000000400cb3 in buddy_free (addr=0x0) at 4a.c:23
23      *node = 0;

(gdb) bt
#0  0x0000000000400cb3 in buddy_free (addr=0x0) at 4a.c:23
#1  0x0000000000400b79 in main () at test_4a.c:81

(gdb) quit
```

Stacktrace (Zurückverfolgung)

Buddy-Verfahren

- Speicherplatzierungsstrategie
- Speicherverwaltung in Blöcken der Größe 2^n
- Sukzessives Halbieren des freien Speichers ($2^n \rightarrow 2^{n-1}$) bis zum „best-fit“ der angeforderten Speichermenge

Buddy-Verfahren: Reservierung

- Suche nach einem Speicherbereich, der die passende Größe hat (minimaler Block mit der Größe $2^k \geq$ angeforderter Speicher)
  - wird ein Speicherbereich der Größe 2^n gefunden → reservieren, Ende
  - sonst versuche diesen wie folgt zu erzeugen:

1. teile einen freien Speicherbereich > 2^n (kleinstmöglich!) in zwei Hälften
2. ist eine Hälfte von der Größe 2^n (oder die untere Grenze erreicht) → reservieren, Ende
3. gehe zu 1.

Buddy-Verfahren: Reservierung

- Suche nach einem Speicherbereich, der die passende Größe hat (minimaler Block mit der Größe $2^k \geq$ angeforderter Speicher)
  - wird ein Speicherbereich der Größe 2^n gefunden → reservieren, Ende
  - sonst versuche diesen wie folgt zu erzeugen:

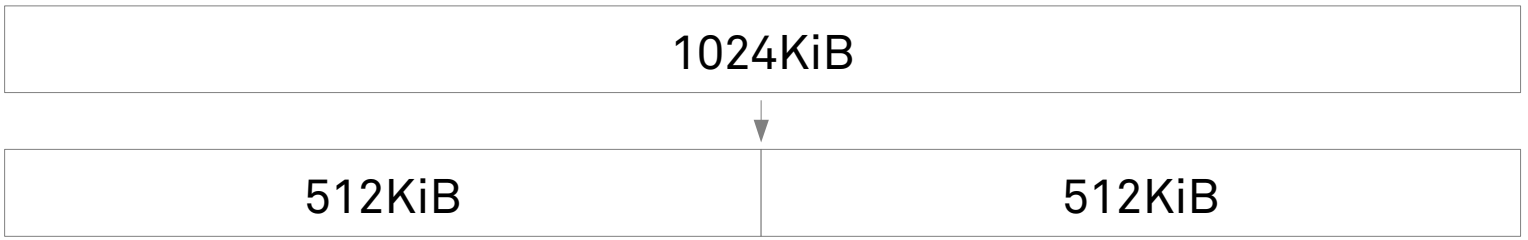
  1. teile einen freien Speicherbereich > 2^n (kleinstmöglich!) in zwei Hälften
  2. ist eine Hälfte von der Größe 2^n (oder die untere Grenze erreicht) → reservieren, Ende
  3. gehe zu 1.
- Bsp.: Anforderung von 200KiB (auf 2^n aufgerundet: 256KiB)

Buddy-Verfahren: Reservierung

- Suche nach einem Speicherbereich, der die passende Größe hat (minimaler Block mit der Größe $2^k \geq$ angeforderter Speicher)
  - wird ein Speicherbereich der Größe 2^n gefunden → reservieren, Ende
  - sonst versuche diesen wie folgt zu erzeugen:

1. teile einen freien Speicherbereich > 2^n (kleinstmöglich!) in zwei Hälften
2. ist eine Hälfte von der Größe 2^n (oder die untere Grenze erreicht) → reservieren, Ende
3. gehe zu 1.

- Bsp.: Anforderung von 200KiB (auf 2^n aufgerundet: 256KiB)



Buddy-Verfahren: Reservierung

- Suche nach einem Speicherbereich, der die passende Größe hat (minimaler Block mit der Größe $2^k \geq$ angeforderter Speicher)
  - wird ein Speicherbereich der Größe 2^n gefunden → reservieren, Ende
  - sonst versuche diesen wie folgt zu erzeugen:

  1. teile einen freien Speicherbereich > 2^n (kleinstmöglich!) in zwei Hälften
  2. ist eine Hälfte von der Größe 2^n (oder die untere Grenze erreicht) → reservieren, Ende
  3. gehe zu 1.
- Bsp.: Anforderung von 200KiB (auf 2^n aufgerundet: 256KiB)

Buddy-Verfahren: Freigabe

- Gebe den Speicherbereich frei und betrachte den angrenzenden Buddy
  - ist dieser ebenfalls nicht belegt, so fasse diese beiden zusammen
  - wiederhole die Zusammenfassung von Buddies, bis ein Speicherbereich belegt oder der ganze Speicher freigeben ist

Buddy-Verfahren: Freigabe

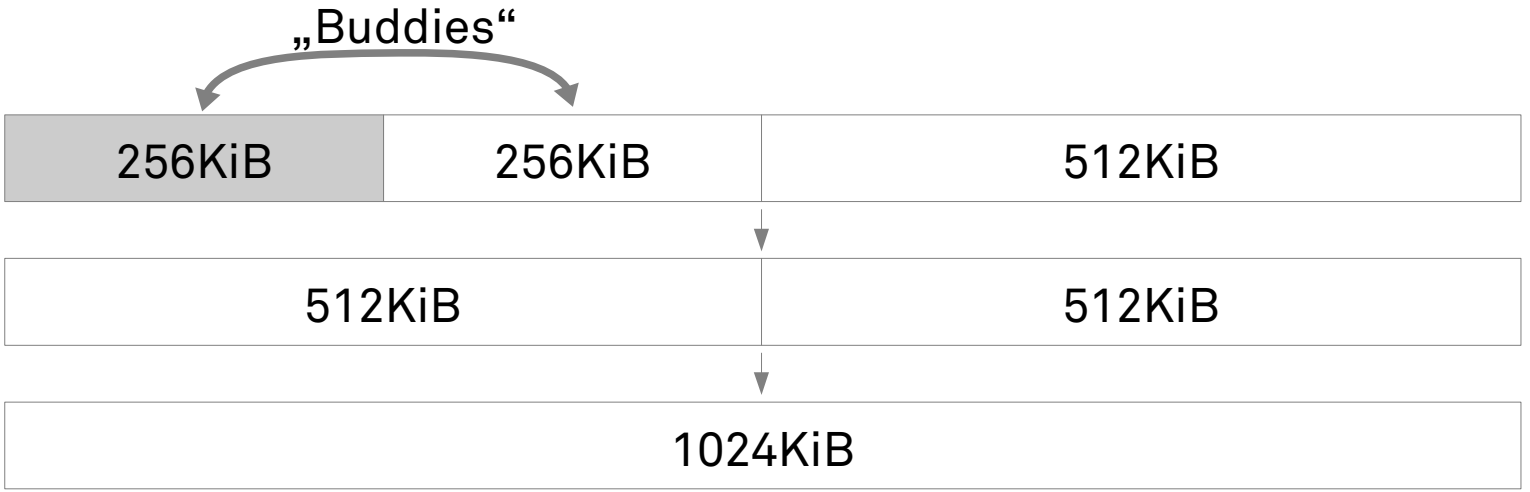
- Gebe den Speicherbereich frei und betrachte den angrenzenden Buddy
  - ist dieser ebenfalls nicht belegt, so fasse diese beiden zusammen
  - wiederhole die Zusammenfassung von Buddies, bis ein Speicherbereich belegt oder der ganze Speicher freigeben ist

Buddy-Verfahren: Freigabe

- Gebe den Speicherbereich frei und betrachte den angrenzenden Buddy
  - ist dieser ebenfalls nicht belegt, so fasse diese beiden zusammen
  - wiederhole die Zusammenfassung von Buddies, bis ein Speicherbereich belegt oder der ganze Speicher freigeben ist

Buddy-Verfahren: Freigabe

- Gebe den Speicherbereich frei und betrachte den angrenzenden Buddy
  - ist dieser ebenfalls nicht belegt, so fasse diese beiden zusammen
  - wiederhole die Zusammenfassung von Buddies, bis ein Speicherbereich belegt oder der ganze Speicher freigeben ist

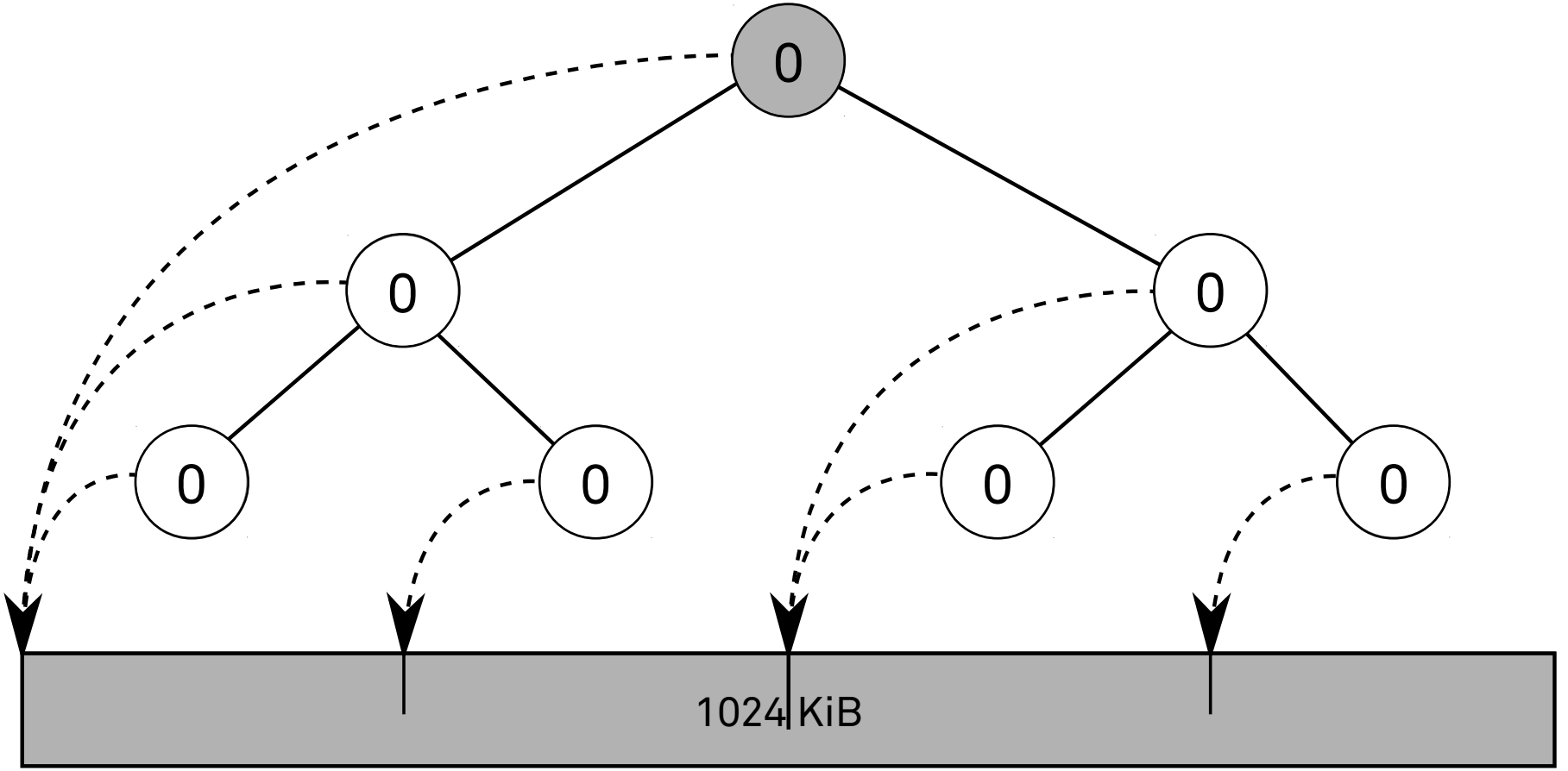


Aufgabe 4 – Speicherverwaltung

- Implementierung des Buddy-Algorithmus
  - Freigabe von Speicher
  - Reservierung von Speicher
  - Effiziente Speicherfreigabe (★)
- Verwendung eines Binärbaums als effiziente Repräsentation
  - Ein Knoten stellt einen Speicherbereich dar
  - Halbierung des Speicherbereichs wird durch Ebenen des Baums repräsentiert
  - Knotenmarkierungen stellen Zustand des Speicherbereichs dar

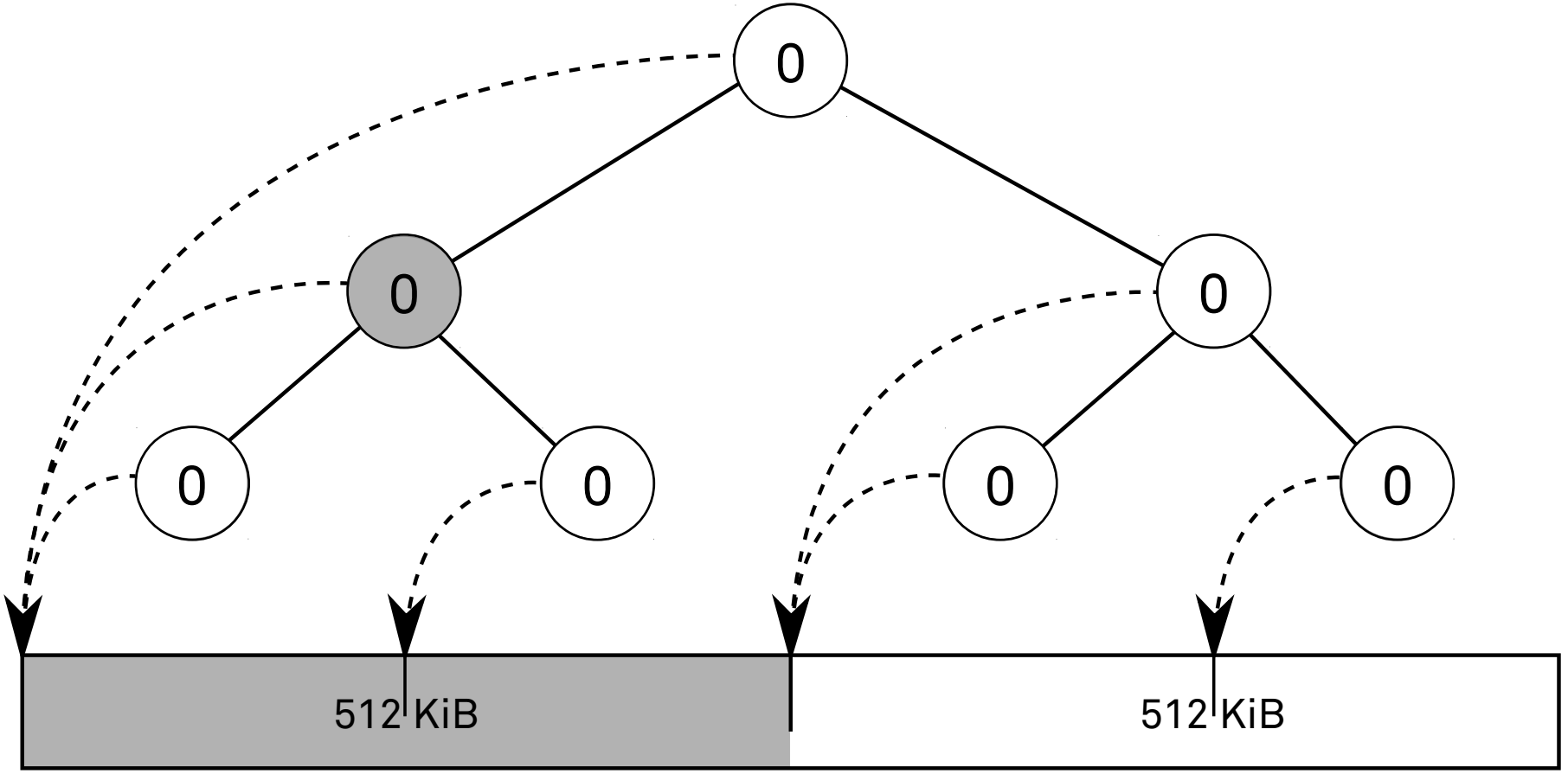
Aufgabe 4: Beispiel Speicherallokation

- Ziel: Allokation von 256 KiB



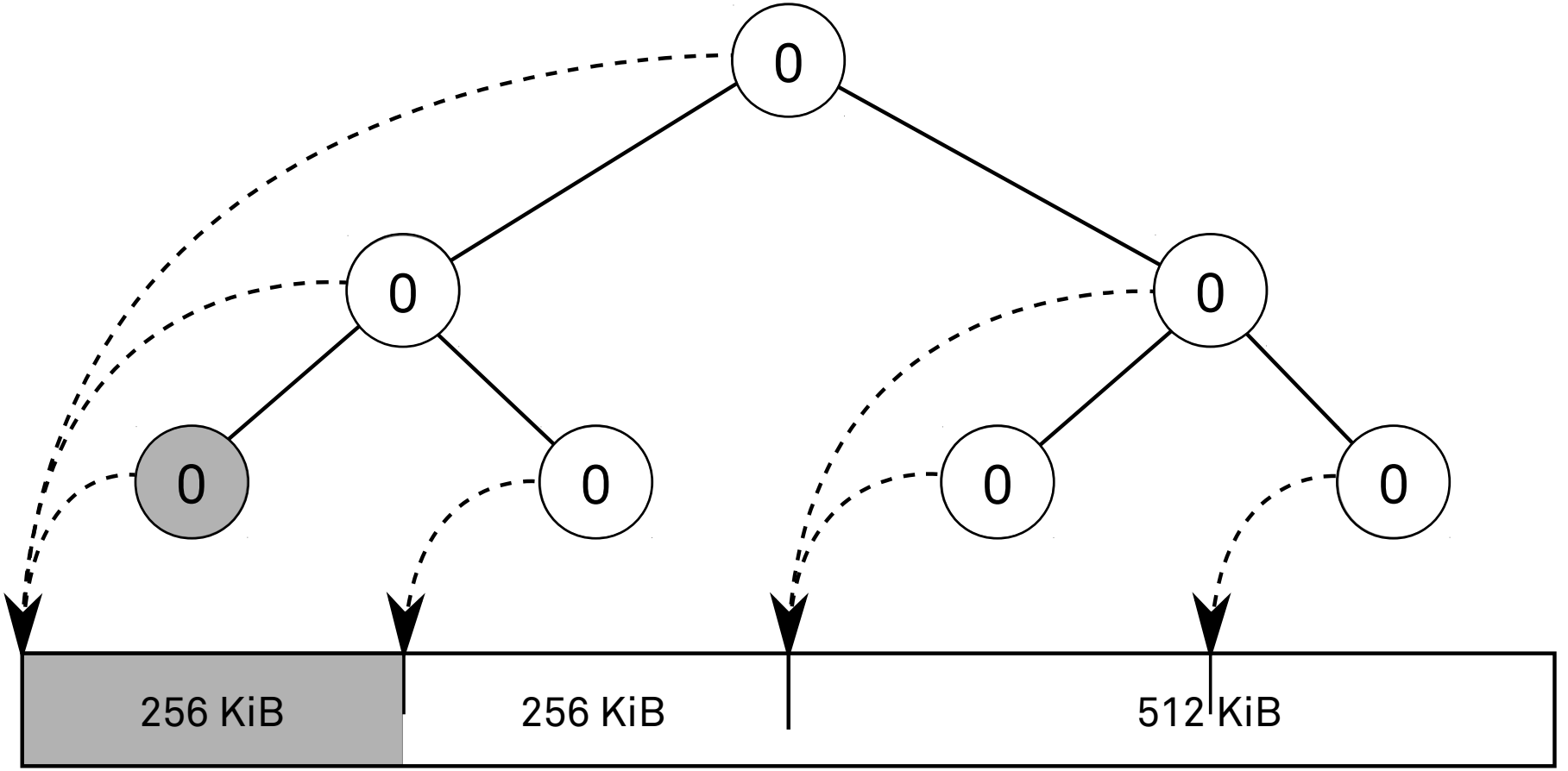
Aufgabe 4: Beispiel Speicherallokation

- Ziel: Allokation von 256 KiB



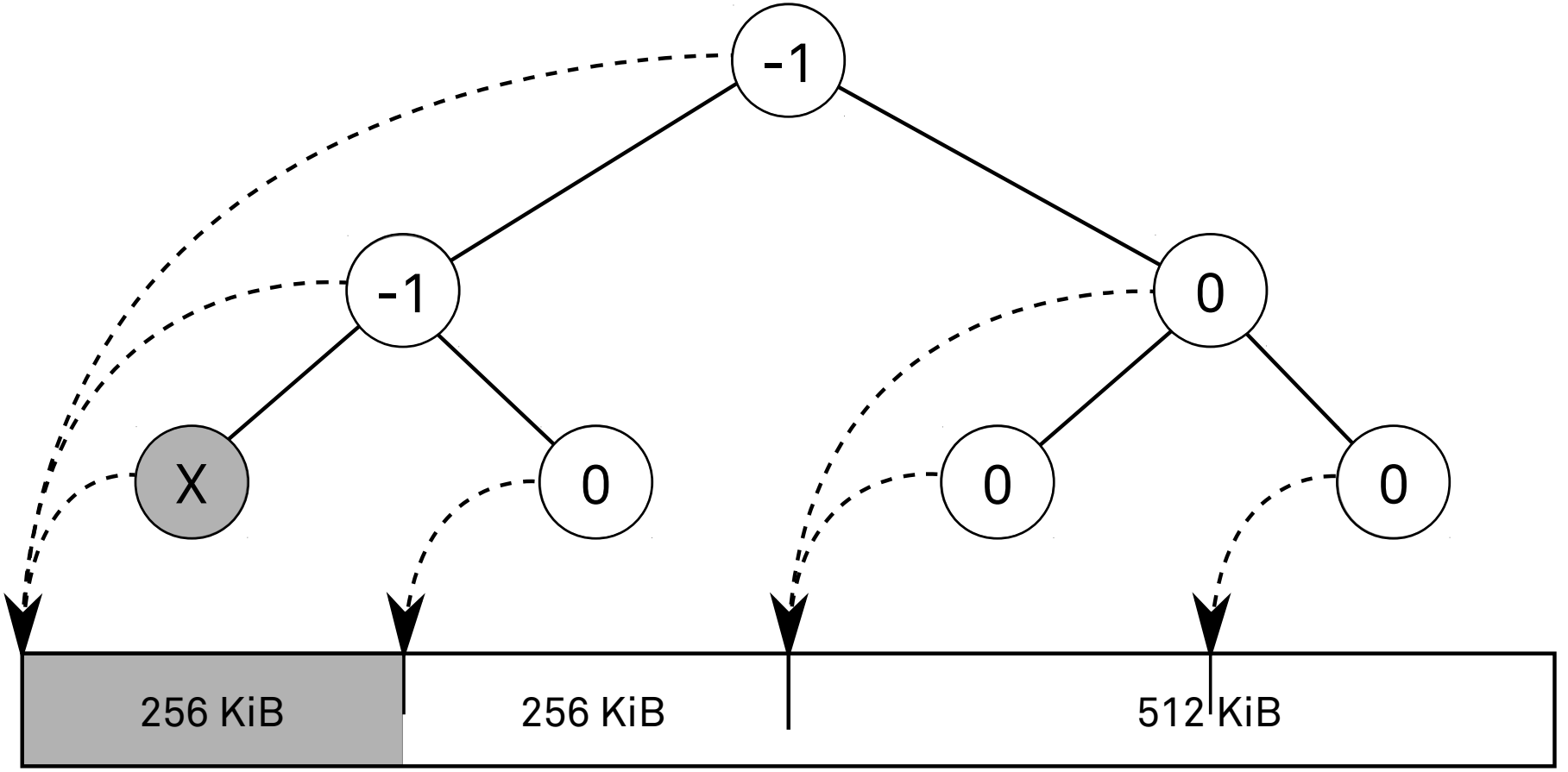
Aufgabe 4: Beispiel Speicherallokation

- Ziel: Allokation von 256 KiB



Aufgabe 4: Beispiel Speicherallokation

- Ziel: Allokation von 256 KiB



Tests in C

- Einfache Programme können bei der Ausführung von Hand getestet werden
  - z.B. „id“: Wird die richtige UID und GID ausgegeben?
- Komplexe Algorithmen und Datenstrukturen benötigen interne Tests

```
// Wurzel des Binärbaums als belegt markieren
bst[0] = ‘A’;
// … und freigeben
buddy_free(…);

// Sie sollte nun als frei markiert sein
if (bst[0] != NODE_FREE) {
    printf(“Da ging wohl etwas schief!”);
    exit(1);
}
```

- Ständige Folge von if–printf–exit ist umständlich und repetitiv

Tests in C: Assertions

- `assert(3)` nimmt diese Arbeit ab

```
// Wurzel des Binärbaums als belegt markieren
bst[0] = ‘A’;
// … und freigeben
buddy_free(…);

// Sie sollte nun als frei markiert sein
assert(bst[0] == NODE_FREE);
printf(“Keine Fehler erkannt\n”);
```

- Ausgabe von Dateiname, Zeile und Test im Fehlerfall

```
derf@vatos A4$ ./test_4a
test_4a: test_4a.c:77: int main(): Assertion `bst[0] == NODE_FREE' failed.
```

- Tests lassen sich per Präprozessor deaktivieren

```
derf@vatos A4$ gcc -Wall -DNDEBUG -o test_4a …
derf@vatos A4$ ./test_4a
Keine Fehler erkannt
```

Klausuraufgabe: Buddy-Verfahren

Platzierungsstrategien (6 Punkte) Die folgenden Tabellen zeigen die Belegung eines Speichers der Größe 32 MiB. In der ersten Tabelle sind beispielsweise 4 MiB von Prozess A belegt. Ergänzen Sie die folgenden Tabellen um Markierungen für die angegebenen Anfragen.

0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30
A	A					B	B					C	C	C	C

Prozess D belegt 6 MiB

Klausuraufgabe: Buddy-Verfahren

Platzierungsstrategien (6 Punkte) Die folgenden Tabellen zeigen die Belegung eines Speichers der Größe 32 MiB. In der ersten Tabelle sind beispielsweise 4 MiB von Prozess A belegt. Ergänzen Sie die folgenden Tabellen um Markierungen für die angegebenen Anfragen.

0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30
A	A					B	B					C	C	C	C

Prozess D belegt 6 MiB

0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30
A	A					B	B	D	D	D	D	C	C	C	C

Klausuraufgabe

Platzierungsstrategien (6 Punkte) Die folgenden Tabellen zeigen die Belegung eines Speichers der Größe 32 MiB. In der ersten Tabelle sind beispielsweise 4 MiB von Prozess A belegt. Ergänzen Sie die folgenden Tabellen um Markierungen für die angegebenen Anfragen.

0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30
A	A			B	B							C	C	C	C

Prozess E belegt 9 MiB

Klausuraufgabe

Platzierungsstrategien (6 Punkte) Die folgenden Tabellen zeigen die Belegung eines Speichers der Größe 32 MiB. In der ersten Tabelle sind beispielsweise 4 MiB von Prozess A belegt. Ergänzen Sie die folgenden Tabellen um Markierungen für die angegebenen Anfragen.

0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30
A	A			B	B							C	C	C	C

Prozess E belegt 9 MiB

0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30
A	A	-	-	B	B	-	-	-	-	-	-	C	C	C	C