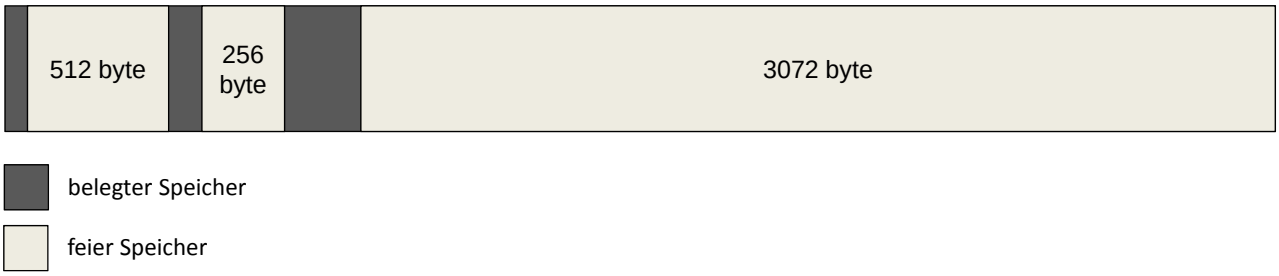
# Übung 7

Abgabe: 28.06.2018 8:30

## Aufgabe 7.1: Speicherbelegung bei Segmentierung (2 + 3 + 1.5 = 6.5 Punkte)

Gegeben sei folgende Belegung eines Speichers:




Zur Belegung des Speichers wird Segmentierung verwendet und es seien vier Belegungsstrategien für Speicherplatzanforderungen aus der Vorlesung gegeben:

| | |
|---|---|
| First-Fit | Belege von vorne beginnend den ersten freien Speicherbereich, der groß genug ist, die Anforderung zu erfüllen. |
| Rotating-First-Fit | Wie First-Fit, jedoch wird vom Ende des zuletzt ausgewählten Speicherbereichs aus ein passender Bereich gesucht. D.h., wurde bei obigem Speicherzustand eine Anfrage nach z.B. 400 Byte in der ersten Lücke von 512 Byte Größe platziert, entsteht zwar eine neue Lücke von 112 Byte Länge, diese wird allerdings bei der nächsten Anfrage übersprungen und die Suche am Beginn der Lücke von 256 Byte Größe fortgesetzt. Wird das Ende des Speichers erreicht, so wird die Suche am Anfang des Speichers fortgesetzt. Bei der ersten Anforderung beginnt die Suche am Anfang des Speichers. |
| Best-Fit | Belege den kleinsten freien Speicherbereich, in den die Anforderung passt. |
| Worst-Fit | Belege den größten freien Speicherbereich, in den die Anforderung passt. |

a) Wie wird der obige Speicher belegt, wenn nacheinander vier Anforderungen der Größe 123 Byte, 389 Byte, 512 Byte und 2271 Byte gestellt werden? Notieren Sie für jede der vier Strategien die freien Speicherbereiche nach jeder Anforderung (z.B. in der Form (512, 256, 3072) für die obige Ausgangsbelegung) und geben Sie an, für welche Strategien die Anforderungen erfüllt werden können.

   Beachten Sie, dass die Daten jeweils linksbündig in einer Lücke abgelegt und einmal belegte Speicherbereiche nicht wieder freigegeben werden.

b) Sie dürfen nun die Reihenfolge der freien Speicherbereiche und die der Anforderungen vertauschen (aber keine freien Speicherbereiche zusammenfassen).

   - Geben Sie eine Anforderungs- und eine Speicherbereichsreihenfolge an, bei der alle vier Strategien zum Erfolg führen.
   - Geben Sie eine Anforderungs- und eine Speicherbereichsreihenfolge an, bei der je First-Fit und Worst-Fit die selbe und Rotating-First-Fit und Best-Fit die selbe Speicherbelegung nach jeder einzelnen Anforderung aufweisen, d.h. die Strategien (First-Fit, Worst-Fit) und (Rotating-First-Fit, Best-Fit) sollen in jedem Schritt den selben Speicherbereich auswählen.

– Geben Sie zusätzlich eine Anforderungs- und eine Speicherbereichsreihenfolge an, bei der nur Rotating-First-Fit und Best-Fit erfolgreich sind.

Geben Sie immer auch die Speicherbelegung nach Abarbeitung der einzelnen Anforderungen an.

c) **Quick-Fit** benutzt verchiedene Bereiche für verschiedengroße Anforderungen. In dieser Teilaufgabe benutzen wir den 1 Block (512 byte) für Anforderungen der Größe 256 byte, den zweiten Block (256 byte) für Anforderungen der Größe 128 byte, und den dritten Block (3072 byte) für Anforderungen der Größe 1024 byte. Anforderungen anderer Größen werden vorher auf die nächstgrößere Größe aufgerundet, sodass Speicher durch Fragmentierung ungenuztzt bleibt. Größere Anforderungen können nicht erfüllt werden.

**Gegeben seien die folgenden Sequenzen von Anforderungen. Sind diese Erfüllbar? Begründen sie jeweils warum diese Erfüllbar oder nicht Erfüllbar sind.**

– (768, 513, 512, 257)
– (1, 2, 129, 128)
– (389, 512, 2271, 123)

## Aufgabe 7.2: Speicherverwaltung mit Buddy-Systemen (4 + 1 + 1 = 7 Punkte)

Gegeben sei ein Rechner, der mit 32 MByte Hauptspeicher ausgerüstet ist. Dieser Speicher wird mit einem Buddy-System verwaltet und ist im Moment leer.

Die Speicherbelegung kann mit Hilfe eines Binärbaumes dargestellt werden. Dabei stellt die Wurzel den gesamten Speicher von 32 MByte dar, die nächste Ebene zwei Speicherbereiche der Größe 16 MByte, usw. Ist der Speicher leer, existiert nur die Wurzel. Bei einer Speicheranforderung wird der Baum zunächst von links nach rechts durchsucht, ob ein passender Speicherbereich existiert. Ein Speicherbereich der Größe $2^x$ ist dann passend, wenn für die Speicheranforderung der Größe $s$ gilt: $2^{x-1} < s \leq 2^x$. Existiert kein solcher Speicherbereich, wird nach einem nächstgrößeren Speicherbereich (der Größe $2^{x+1}$) gesucht. Falls kein solcher existiert, wird wieder nach einem nächstgrößeren gesucht (Größe $2^{x+2}$) usw., bis ein Speicherbereich gefunden oder die Wurzel erreicht wird. Existieren zwei oder mehr gleich große Speicherbereiche, wird immer der am weitesten links im Baum liegende Bereich ausgewählt. Wurde ein Speicherbereich der Größe $2^{x+1}$ oder größer ausgewählt, wird er in zwei Bereiche (Buddies) geteilt und der linke Bereich ausgewählt. Ist er immer noch größer als benötigt, wird er weiter geteilt, so lange bis durch die letzte Teilung zwei Buddies der Größe $2^x$ entstehen; der linke Buddy wird der Anfrage zugeteilt. Wird kein Speicherbereich gefunden, der groß genug ist, wird die Suche erfolglos abgebrochen. Bei Freigabe von Speicherbereichen werden benachbarte freie Buddies so weit möglich zu einem einzigen freien Speicherbereich zusammengefasst.

Der Reihe nach werden nun Speicherbereiche folgender Größe angefordert bzw. freigegeben:

| Nr. | Operation | Name | Größe in MByte |
|---|---|---|---|
| 1 | Anforderung | A | 5 |
| 2 | Anforderung | B | 9 |
| 3 | Anforderung | C | 1 |
| 4 | Anforderung | D | 2 |
| 5 | Freigabe | A | |
| 6 | Freigabe | C | |
| 7 | Anforderung | E | 3 |
| 8 | Anforderung | F | 4 |

a) Stellen Sie die Speicherbelegung nach jeder Anforderung bzw. Freigabe als Binärbaum dar. Vor der ersten Anforderung sei der Speicher leer, d.h. es existiert nur die Wurzel, die den gesamten Speicher von 32 MByte darstellt. Existieren mehrere passende Speicherbereiche für eine Anforderung, werde immer der am weitesten links im Baum liegende Bereich ausgewählt.

b) Gegeben sei die Speicherbelegung nach ausführung aller acht Schritte. Wie groß ist der größte zusammenhängende freie Speicherbereich? Was ist die größmöglichste Speicheranforderung, die noch gewährt werden kann? Wieso ist es bei einem Buddy System möglich, dass diese beiden größen sich unterscheiden?

c) Interne Fragmentierung entsteht, wenn ein zugeteilter Speicherblock nur zum Teil gefüllt wird. Welcher Anteil des gesamten Speichers geht durch interne Fragmentierung verloren?

### Aufgabe 7.3: Adressen (1 + 2 = 3 Punkte)

Für die Speicherverwaltung nach dem Segmentierungsverfahren sei für einen Prozess die folgende Segmenttabelle gegeben:

| Segment | Basisadresse | Länge |
|---|---|---|
| 0 | 51 | 999 |
| 1 | 1850 | 45 |
| 2 | 1895 | 410 |
| 3 | 1400 | 175 |
| 4 | 2500 | 191 |
| 5 | 4400 | 53 |
| 6 | 4107 | 65 |

Sowohl Basisadresse als auch Länge seien in Byte angegeben.

a) Wieviele Byte stehen dem Prozess im physikalischen Speicher zur Verfügung?

b) Berechnen Sie zu den folgenden physikalischen Adressen jeweils die logischen Adressen. (Eine logische Adresse wird hier durch das Paar (Segment-Nummer, Offset) beschrieben.), oder geben sie an, dass dieser physikalischen Addresse keine logische Addresse zugeordnet ist.

1. 1895
2. 1575
3. 1935
4. 1050

### Aufgabe 7.4: Page Table (1 + 0,5 + 1 = 2,5 Punkte)

Gegeben sei ein Betriebssystem, das Paging mit einer Seitengröße von 4 KiB einsetzt. Die logischen Adressen der Prozesse sind 32 Bit lang.

a) Es wird eine einstufige Page Table verwendet; wie viele Einträge hat die Page Table? Geben Sie an, wie Sie auf Ihr Ergebnis kommen.

b) Nun wird das System verändert: es wird zweistufiges Paging eingesetzt, wobei die äußere und die inneren Page Tables gleich groß sind. Welchen Umfang haben die Tabellen nun?

c) Welchen Vorteil hat mehrstufiges Paging? Welchen Nachteil hat es? Begründen Sie Ihre Aussagen knapp.

### Aufgabe 7.5: Shared Pages (0,5 + 0,5 + 0,5 + 0.5 = 2 Punkte)

**Beantworten sie die folgenden Fragen jeweils mit Begründung.**

a) Warum ist es sinvoll Page Frames zwischen mehreren Prozessen zu sharen?

b) Haben Pages Frames die zwischen mehreren Prozessen geshared werden, in allen Prozessen die selbe logische Adresse?

c) Benutzen Prozesse die Page Frames sharen eine identische Page Table?

d) Bei Copy-on-Write werden Pages die zwischen mehreren Prozessen geshared werden auf read-only gesetzt. Erst wenn ein Prozess versucht in diese Page zu schreiben wird der Inhalt in ein neues Frame kopiert und die Pages auf read/write protection gesetzt. Welchen Vorteil hat Copy-on-Write bei der Erzeugung eines Kindprozesses mittels `fork()`?