Betriebs- und Kommunikationssysteme

WS 15/16

Prof. Dr-Ing. Volker Roth

Freie Universität Berlin

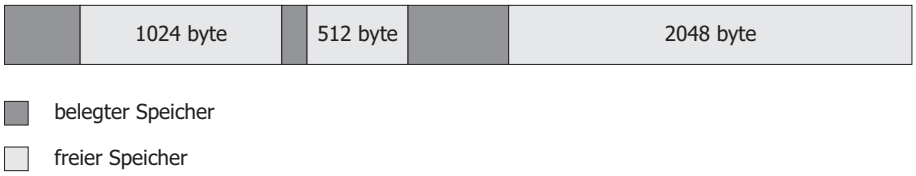
6. Übung

Ausgabe	Abgabe
20.11.15	4.12.15

Bitte bei der Abgabe Name der Mitglieder einer Gruppe, Nummer der Übung/Teilaufgabe und Datum auf den Lösungsblättern nicht vergessen! Darauf achten, dass die Lösungen beim richtigen Tutor abgegeben werden. Achten Sie bei Programmieraufgaben außerdem darauf, dass diese im Linuxpool kompilierbar sind. Nutzen Sie dazu die Flags `-std=c99`, `-Wall` und `-pedantic`. Es sollten keine Warnungen auftauchen.
Zu spät abgegebene Lösungen werden nicht berücksichtigt!

Aufgabe 1: Speicherverwaltung (2 Punkte)

(a) Gegeben sei folgende Belegung eines Speichers:



Weiterhin seien drei Belegungensmethoden für Speicherplatzanforderungen 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 von der Position der vorherigen Platzierung ausgehend ein passender Bereich gesucht. Wird das Ende des Speichers erreicht, so wird die Suche am Anfang des Speichers fortgesetzt (maximal bis zur Position der vorherigen Platzierung). Bei der ersten Anforderung beginnt die Suche am Anfang des Speichers.

Best-Fit: Belege den kleinsten freien Speicherbereich, in den die Anforderung passt.

(i) Wie sieht der obige Speicher aus, wenn nacheinander vier Anforderungen der Größe 300 Byte, 512 Byte, 2048 Byte und 624 Byte ankommen? Notieren Sie für jede Strategie die freien Speicherbereiche nach jeder Anforderung (z.B. (1024, 512, 2048) für die obige Belegung), und geben Sie an, für welche Methoden die Anforderungen erfüllt werden können!

(ii) Geben Sie für jede Strategie, die oben nicht alle Anforderungen erfüllen konnte, eine andere Reihenfolge der obige Anforderungen, so dass die Strategie alle Anforderungen erfüllen kann. Notieren Sie wieder die freien Speicherbereiche nach jeder Anforderung.

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

(b) Ein großer Vorteil der virtuellen Speicherverwaltung ist, dass der Programmierer sich nicht um den physikalischen Adressraum kümmern muss, sondern mit logischen Adressen arbeiten kann. Spätestens bei der Ausführung eines Programmes müssen diese logischen Adressen aber in physikalische Adressen umgewandelt werden.

Das Umsetzungsprinzip beim *Segmenting* funktioniert wie folgt: Jedem Programm ist eine Segmenttabelle zugeordnet, in der für jedes logische Segment die physikalische Anfangsadresse `b` und die Größe des Segments `l` eingetragen ist, die das Programmsegment im Speicher einnimmt.

Eine logische Adresse besteht nun aus der Segmentnummer `s`, mit der aus der Segmenttabelle die entsprechende Anfangsadresse `b` bestimmt wird, und einem Offset `d`, der angibt, an welcher Stelle in Relation zur Anfangsadresse sich die Adresse, auf die man zugreifen will, befindet. Dabei wird geprüft, ob der Offset kleiner als die Segmentlänge `l` ist, da anderenfalls eine ungültige Adresse angesprochen wird. Die physikalische Adresse ergibt sich also zu `b+d`.

Für die Speicherverwaltung nach dem Segmentierungsverfahren ist für ein Programm folgende Segmenttabelle gegeben (Länge in Speicherworten):

Segment	Basis	Länge
0	1410	395
1	500	120
2	630	515
3	420	70
4	1145	150

(1) Wie viele Speicherworte stehen dem Programm im physikalischen Speicher zur Verfügung?

(2) Welches ist die kleinste und welches die größte für das Programm verfügbare physikalische Adresse?

(3) Berechnen Sie zu den folgenden physikalischen Adressen jeweils die logischen Adressen:

i. 762
ii. 1145
iii. 1146
iv. 625

(4) Berechnen Sie zu den folgenden logischen Adressen jeweils die physikalischen Adressen:

Segment	Basis
3	23
1	128
4	0
4	1

Aufgabe 2: Speicherverwaltung in C (3 Punkte)

Schreiben Sie eine Speicherverwaltung in C, ohne dabei auf `malloc()` oder `free()` der Standardbibliothek zuzugreifen. Ihre Speicherverwaltung soll die folgenden Funktionen enthalten:

```
1      void* my_malloc(int byteCount);
2      void my_free(void* p);
```

Dabei soll `my_malloc()` einen zusammenhängenden Speicherbereich der Größe `byteCount` reservieren und `my_free` einen auf diese Art reservierten Speicherbereich wieder freigeben. Zur Lösung der Aufgabe können Sie das mm-Framework (MemoryManagement) mit der Testdatei `test_mm.c` benutzen. Sie können diese mit `gcc -std=c99 -Wall -pedantic -o <ausgabe> mm.c test_mm.c` übersetzen und ausführen lassen. Beachten Sie:

- Es sollte jeweils möglichst nur so viel Speicher reserviert werden, wie angefordert wurde.
- Wird ein Speicherbereich freigegeben, so können mehrere freie Speicherbereiche nebeneinander vorliegen. Diese sollten zu einem großen Speicherbereich zusammengefasst werden, da ansonsten eine unnötige Fragmentierung des Speichers vorliegt.

Hinweis:

Das mm-Framework bietet Ihnen ein Gerüst, um die Aufgabe zu lösen. Beachten Sie, dass sie die Verwaltungsstruktur - in diesem Fall eine einfach verkettete Liste - in dem selben "Speicher"' abgelegt werden soll, wie die Daten selbst. Dies kann z.B. wie folgt erreicht werden: Der Speicher wird in Blöcke unterteilt. Dabei hat jeder Block einen Header, in dem Informationen über den Speicherblock enthalten sind. Nach dem Header folgen die eigentlichen Daten, die verwaltet werden. Die einzelnen Blöcke können weiterhin mittels ihrem Header auf andere Blöcke (einfach verkette Liste) verweisen: `[BlockHeader|Daten]` → `[BlockHeader|Daten]` → `[BlockHeader|Daten]`. Die folgende Ausgabe der status()-Funktion beschreibt einen solchen möglichen Aufbau.

```
1      Uebersicht des Speichers: 9996 / 10240 Speicher frei
2
3      # at allocated space data next block
4      1 0x8049140 TRUE 4 [0x8049150,0x8049153] 0x8049154
5      2 0x8049154 TRUE 80 [0x8049164,0x80491b3] 0x80491b4
6      3 0x80491b4 FALSE 4 [0x80491c4,0x80491c7] 0x80491c8
7      4 0x80491c8 TRUE 80 [0x80491d8,0x8049227] 0x8049228
8      5 0x8049228 FALSE 9992 [0x8049238,0x804b93f] (nil)
```

In diesem Beispiel hat jeder Header eine Größe von 16 Bytes (siehe mm.c).