TI III: Operating and Communication Systems WS 2010/11 Übungsblatt Nr. 4

Dipl.-Inform. Heiko Will, AG Technische Informatik, Freie Universität Berlin

Ausgabe am 12.11.2010 — Abgabe spätestens 19.11.2010, 10:00 Uhr

Bitte bei der Abgabe beide Namen/Matr.Nr. der Mitglieder einer Gruppe, NUMMER DER ÜBUNG/TEILAUFGABE und DATUM auf den Lösungsblättern **nicht vergessen**! Darauf achten, dass die Lösungen beim richtigen Tutor/der richtigen Tutorin abgegeben werden.
Achten Sie bei Programmieraufgaben außerdem darauf, dass diese im Linuxpool kompilierbar sind.
Zu spät abgegebene Lösungen werden nicht mehr berücksichtigt!

---

1. Aufgabe:

Erklären Sie die unterschiedlichen Auswirkungen von externer Fragmentierung bei Festplatten und RAM.

2. Aufgabe:

Ein Programm was direkt nach dem Starten die Adresse der Mainfunktion ausgibt und dann in eine Endlosschleife verfällt, wird dreimal parallel gestartet. Es wird dreimal dieselbe Adresse ausgegeben werden (ausprobieren!). Teilen sich diese drei Instanzen einen gemeinsamen Speicher oder nicht? – Bitte ausführlich begründen, was Betriebssystem und CPU bis zum physikalischen Speicherzugriff für Mechanismen anwenden.

3. Aufgabe:

Wie oft können sie unter Linux malloc(1000000000) erfolgreich aufrufen, wenn ihr System 1GB RAM und ein 2GB Swapfile hat. Recherchieren und Begründen Sie, bzw. probieren Sie es aus!

4. Aufgabe: Programmieren in c.

C Aufgabe: Speicherverwaltung

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

- void* my_malloc(int byteCount);
- 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 -o <ausgabe> mm.c test_mm.c übersetzen und ausführen lassen.
Beachten Sie:

1. Es sollte jeweils möglichst nur so viel Speicher reserviert werden, wie angefordert wurde.
2. 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.

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

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