ulm university universität
uulm

Institut für Angewandte Informationsverarbeitung
Dr. Andreas F. Borchert
Stefan Lenz

6. Dezember 2012
Blatt 7

Systemnahe Software I (WS 2012/2013)

Abgabe bis zum 20. Dezember 2012, 16:00 Uhr

Lernziele:

- Eine Speicherverwaltung selbst entwerfen und programmieren

Aufgabe 11: Eigene Speicherverwaltung (20 Punkte)

Aufgabe dieses zweiwöchigen Blattes ist es, eine eigene Speicherverwaltung zu schreiben. Wir gehen davon aus, dass das System uns nur eine einzige Funktion zur Belegung von Speicher auf dem Heap zur Verfügung stellt:

```
void* get_block_from_system()
```

Diese Funktion liefert einen Zeiger auf einen freien Speicherblock der Größe BLOCKSIZE (8192 Bytes) zurück. Auf `malloc()`, `calloc()` oder Ähnliches dürft Ihr also nicht zurückgreifen!

Auf Basis dieser Funktion sollt Ihr eine Speicherverwaltung zur Belegung kleinerer Speicherbereiche aufbauen, die folgende Funktionen umfassen soll:

- `void init_my_alloc()`

  Wird einmal zu Beginn aufgerufen, um die eigene Speicherverwaltung zu initialisieren. Wenn das nicht nötig ist, muss die Funktion trotzdem (mit leerem Rumpf) implementiert werden.

- `void* my_alloc(size_t size)`

  Liefert einen Zeiger auf einen, vom Aufrufer anschließend frei verwendbaren, Speicherbereich der Größe `size` zurück. Dieser Speicherplatz muss in einem Bereich sein, der vorher vom System mit `get_block_from_system()` geholt wurde.

- `void my_free(void *ptr)`

  Gibt einen vorher mit `my_alloc()` reservierten Speicherbereich wieder frei. Der so freigegebene Speicherplatz kann später von `my_alloc()` wieder neu vergeben werden.

Dabei soll folgendes beachtet werden:

- Ihr könnt davon ausgehen, dass die angeforderte Größe des Speicherbereichs bei einem Aufruf von `my_alloc()` ein Vielfaches von 8 ist und dass kein Speicherblock größer als 256 Byte angefordert wird. Alle vorkommenden Speicherbereiche müssen an einer Adresse, die ein Vielfaches von 8 ist, beginnen.

  Die Funktion `get_block_from_system()` hält sich daran. Bei den von `my_alloc()` zurückgelieferten Adressen ist es eure Aufgabe, dies sicherzustellen.

- Eure Implementierung darf maximal 256 Byte an globalen Variablen verwenden. (Dazu zählen auch Variablen, die innerhalb einer Prozedur als static deklariert wurden.)

- Zu einer vollständigen Lösung gehört auch eine kurze Beschreibung, nach welcher Methode die Speicherverwaltung funktioniert.

- Eine vollständige Lösung sollte mindestens 50 000 Operationen in erträglicher Zeit durchführen können.

- Es sollte nicht mehr als 30 % überschüssiger Speicher vom System geholt werden.

Ihr könnt davon ausgehen, dass `my_alloc()` und `my_free()` korrekt verwendet werden. Das bedeutet, dass nur Speicher freigegeben wird, der vorher auch belegt wurde, dass die angeforderte Größe ein Vielfaches von 8 und nicht größer als 256 ist, etc. Es muss natürlich damit gerechnet werden, dass der mit `my_alloc()` geholte Speicher auch tatsächlich verwendet wird.

Ladet die Testumgebung von der Vorlesungshomepage herunter und bearbeitet die Datei `my_alloc.c` entsprechend den obigen Anforderungen.

Diesmal gibt es zwei verschiedene, *sich gegenseitig ausschließende Methoden*, das Übungsblatt einzureichen:

Für Alle, die die ganze Vorlesung im Umfang von 6 LP hören wollen:

```
submit ss1 11 my_alloc.c team
```

Ab hier brauchen nur noch die Physiker, die für 2 LP ein Testat ablegen wollen, weiter lesen!

Euer Submit-Befehl ist:

```
submit ss1 12 my_alloc.c team
```

Wendet Euch bitte selbständig an die Tutoren Fabian Berstecher und Michael Thoma, um einen Termin für die Testate zu vereinbaren. Die Termine werden im Zeitraum vom 17. bis zum 20. Dezember sein. Jede Gruppe soll dann Ihre Lösung präsentieren können und Fragen zum Hintergrund beantworten.

Die Anmeldung im Hochschulportal kann/muss erst am Ende der Vorlesungszeit des Semesters erfolgen.

Viel Erfolg!