Betriebssysteme
bs@ibr.cs.tu-bs.de
Wintersemester 2025/26

Technische Universität Braunschweig

Übungsblatt 2: Dynamische Speicherverwaltung

Abgabetermin: 21.11.2025 23:59 Uhr

Allgemeine Hinweise zu den BS Übungen

- Die Aufgaben sind alleine zu bearbeiten, allerdings ist Partnerarbeit (maximal zu zweit) möglich. Beide Partner müssen aber in ihrem Git-Repository abgeben und eine `PARTNER`-Datei mit dem Name des Partners im Repository anlegen.
- Jeder Teilnehmende muss dabei in der Lage sein, die gesamte Lösung zu erläutern.
- Zur Versionsverwaltung wird jedem Teilnehmenden ein Git-Repository zur Verfügung gestellt, welches auch zur Abgabe verwendet wird. Der Zugriff auf das Repository erfolgt über das IBR GitLab unter `https://gitlab.ibr.cs.tu-bs.de`.
- Die Abgabe ist spätestens zur Deadline durch ein `git push` zu tätigen. Es sollte sichergestellt werden, dass die Änderungen vorher auch durch ein `git commit` gesichert wurden.
- Eine Aufgabe kann beliebig oft abgegeben werden; es gilt die letzte *rechtzeitige* Abgabe (der letzte rechtzeitige Commit auf dem `main` Branch).
- Die abgegebenen Programme werden auf Ähnlichkeit mit anderen Programmen desselben Semesters und früherer Semester überprüft. Bei starken Übereinstimmungen behalten wir uns ein Ausschluss vom Übungsbetrieb und eine Meldung an den Prüfungsausschuss vor.
- Die Lösungen der Programmieraufgaben müssen im GitLab CI/CD lauffähig sein. Zusätzlich kann das System `x1.ibr.cs.tu-bs.de` zum Testen genutzt werden. Auf diesem können Sie sich per SSH mit ihrem IBR Account anmelden.
- Für die Bearbeitung kann auf einem beliebigen Linux-System gearbeitet werden. Allerdings empfehlen wir das frühzeitige Testen der Abgabe durch ein `git push` und die Inspektion der Test Pipeline im GitLab.
- Die Programmieraufgaben werden automatisch durch das bereitgestellte Testsystem per GitLab CI/CD ausgewertet. Es müssen mindestens $50\,\%$ der verfügbaren Testcase-Punkte erreicht werden.
- Die Testcase-Punkte können kontinuierlich über die Ausgabe des `run-tests` Jobs aus der GitLab CI/CD Pipeline eingesehen werden. Gewertet wird der letzte Pipeline-Status auf dem von uns zur Abgabe angelegten Abgabebranch (z.B. `abgabe1`), überprüfen Sie diesen gegebenenfalls.
- Sollten Sie $50\,\%$ der Punkte erreicht haben werden Sie ggf. für ein Code-Interview ausgewählt. Sollten Sie zum Code-Interview nicht erscheinen oder ihre Lösung nicht ausreichend erklären können wird der Zettel als „ungenügend“ bewertet.
- **KI Richtline**: Der Einsatz von KI-gestützten Tools ist bei der Bearbeitung der Hausaufgaben streng untersagt, da die eigenständige Erarbeitung der Lösungen inhärentes Lernziel ist. Es ist zu beachten, dass wir alle Abgaben als Prüfungsdokumente archivieren und es auch bei einer nachträglichen Prüfung zu einer rückwirkenden Aberkennung der Studienleistung kommen kann.

Aufgabe 1: Programmieren mit Zeigern – Freispeicherverwaltung

In dieser Aufgabe soll eine einfache Freispeicherverwaltung implementiert werden, welche die Funktionen **malloc(3p)**, und **free(3p)** aus der Standard-C-Bibliothek ersetzt. Die Verwaltung des Speichers der Größe $1\,\mathrm{MiB}$ erfolgt mit Hilfe einer einfach verketteten Liste. Im Gegensatz zur lilo-Aufgabe, werden die Listenelemente nun aber nicht per `malloc` angefordert, sondern direkt in den vorgegebenen Speicherbereich gelegt. Die einzelnen Listenelemente, die die Größe des verwalteten Speicherbereichs beinhalten, werden jeweils am Anfang des dazugehörigen Speicherbereiches abgelegt.

(a) Die Funktion `halde_malloc` sucht in der Freispeicherliste den *ersten* Speicherbereich (first-fit Strategie), der für die angeforderte Speichermenge groß genug ist, und entfernt ihn aus der Freispeicherliste. Ist der Speicherbereich größer als benötigt und verbleibt *genügend* Rest, so wird dieser Speicherbereich geteilt und der Rest wird mit Hilfe eines neuen Listenelementes in die Freispeicherliste eingehängt. Im herausgenommenen Listenelement wird statt eines *next*-Zeigers eine *Magic Number* mit dem Wert `0xbaadf00d` eingetragen. Der von `halde_malloc` zurückgelieferte Zeiger zeigt auf die Nutzdaten hinter dem Listenelement.

(b) Die Funktion `halde_free` hängt den freizugebenden Speicherbereich wieder an der richtigen Stelle, sortiert nach ihrer Position im Speicher, in die Freispeicherliste ein. Vor dem Einhängen muss die *Magic Number* überprüft werden. Schlägt die Überprüfung fehl, so soll das Programm durch den Aufruf der Funktion **abort(3)** abgebrochen werden. Für diesen ersten Aufgabenteil kann das Verschmelzen von benachbarten freien Bereichen unterbleiben.

(c) Zu Beginn dieser Teilaufgabe sollte die Testfälle `02_malloc.test` und `03_free.test`[1] erfolgreich bis zum Ende durchlaufen. Bei zunehmender Benutzung unseres Allokators fragmentiert der Speicher in der Freispeicherliste zunehmend. Implementieren Sie daher das verschmelzen benachbarter freier Speicherbereiche. Dieses verschmelzen soll in `halde_free` stattfinden. Sorgen Sie dafür, dass die Freispeicherliste immer anhand der Anfangsaddressen der Speicherbereiche sortiert bleibt. Nach der Einfügeoperation in die Freispeicherliste, können Sie sodann alle freien Nachbarn die am Einfügeort entstanden sind verschmelzen.

Bitte bearbeiten Sie diesen Aufgabenteil, gemäß der Vorgabe in der Datei `halde.c`.

Unvollständige Checkliste:

Die folgenden Punkte der *unvollständigen* Checkliste sollten Sie vor der finalen Abgabe mindestens abarbeiten:

- Die Funktionen `halde_malloc` und `halde_free` weisen das in den Manual-Pages von **malloc(3p)** und **free(3p)** beschriebene Verhalten auf, auch in den genannten Grenzfällen (z.B., `free(NULL)`).
- Die **errno(3)** wird im Fehlerfall korrekt gesetzt.
- Die Anforderung eines Speicherbereiches der Größe (*1 MiB - Größe eines Listenelementes*) ist erfolgreich.

Hinweise:

- Nützliche Manual-Pages: **abort(3)**, **free(3posix)**, **malloc(3posix)**.
- Die Funktion `halde_print` gibt für jedes Listenelement die Position im Adressraum (`addr`), den Offset innerhalb der 1 MiB (`offset`) und die eingetragene Größe (`size`) auf den Standardfehlerkanal aus.
- Bestimmte Testcases laufen nur durch wenn die entsprechenden Marker (Kommentare im Source Code) entfernt wurden. Denken Sie daran diese Marker zu löschen sobald Sie die erforderliche Funktionalität implementiert haben.
- Es gibt dieses mal ein `Makefile` was den Code baut und die Tests ausführen kann. Dafür einfach im Ordner `make` oder `make test` eingeben. Mehr Informationen zu `make` gibt es in der TÜ am 14.11.

[1] `tests/unittest.py -t tests/02_malloc.test`