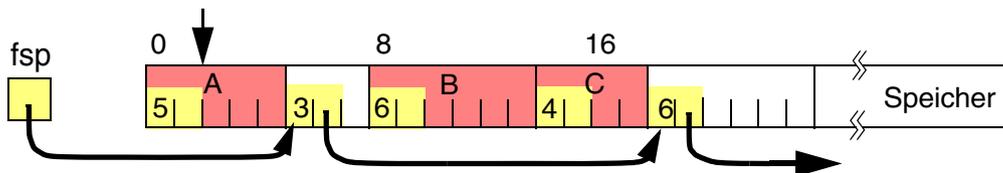


**Aufgabe 4: halde (12 Punkte)** Bearbeitung in Zweier-Gruppen

13.11.2008

In dieser Aufgabe soll eine einfache Freispeicherverwaltung implementiert werden, welche die Funktionen **malloc(3)**, **calloc(3)**, **realloc(3)** und **free(3)** aus der Standard-C-Bibliothek ersetzt. Die Freispeicherverwaltung soll den freien Speicher in einer einfach verketteten Liste verwalten. Am Anfang eines freien Speicherbereichs steht jeweils eine Verwaltungsstruktur mit der Größe des Speicherbereichs und einem Zeiger auf den nächsten freien Bereich. Ein Zeiger (hier `fsp` genannt) zeigt auf den Anfang der Liste.

Im Verzeichnis `/proj/i4sp/pub/aufgabe4/` befinden sich die Dateien `halde.c`, `halde.h`, `malloc-test.c` und `bug.c` sowie ein passendes `Makefile`. Kopieren Sie sich die Dateien in Ihr Arbeitsverzeichnis und implementieren Sie die fehlenden Funktionen und Definitionen in der Datei `halde.c`.

**a) Speicher initialisieren, belegen und freigeben**

Implementieren Sie die Funktionen **malloc** und **free**. Beim ersten Aufruf von **malloc** ist vom Betriebssystem ein Speicherblock von 1 Megabyte (1048576 Bytes) anzufordern (**sbrk(2)**). Der komplette Block wird in die Freispeicherliste eingefügt. Ein Nachfordern von mehr Speicher vom BS ist in dieser Aufgabe **nicht** vorgesehen. Die Funktion **malloc** sucht den ersten freien Speicherbereich in der Freispeicherliste, der für den geforderten Speicherbedarf und die Verwaltungsstruktur groß genug ist (first-fit). Ist der Speicherbereich größer als benötigt und verbleibt *genügend* Rest, so ist dieser Rest mit einer neuen Verwaltungsstruktur am Anfang wieder in die Freispeicherliste einzuhängen. In die Verwaltungsstruktur vor dem belegten Speicherbereich wird die Größe des Bereichs und statt des next-Zeigers eine Magic Number mit dem Wert *Oxcafebab* eingetragen. Der zurückgelieferte Zeiger zeigt hinter die Verwaltungsstruktur, wie in der Abbildung für den Bereich A eingezeichnet. Die Funktion **free** hängt einen freigegebenen Speicher wieder in die Freispeicherliste ein. Vor dem Einhängen ist die Magic Number zu überprüfen, besitzt diese nicht den Wert *Oxcafebab*, so soll das Programm abgebrochen werden (**abort(3)**).

**b) Speicher vergrößern und verkleinern**

Nun sollen noch die Funktionen **realloc** und **calloc** implementiert werden (**memcpy(3)**, **memset(3)**).

**c) Debuggen**

Debuggen Sie nun das Programm **bug** und beschreiben Sie die Fehler, die Sie finden, in der Datei `halde.txt`. Benutzen Sie zum Debuggen einen Debugger wie z.B. `gdb`. Von besonderem Interesse sind Fehler, die zu einem Fehlverhalten in Ihrer Freispeicherverwaltung führen.

**d) SPARC-Variante**

Testen Sie Ihre Lösung auch auf dem Rechner `fau04{a,c,d}`. Diese Rechner haben eine SPARC-CPU und ein Solaris {10, 9, 8} Betriebssystem. Sie müssen deshalb auf diesen Rechnern neu kompilieren und die Binärdateien statt in `bin.i386` in einem Unterverzeichnis `bin.sparc` ablegen.

Das Besondere an der SPARC-Architektur ist, dass - im Gegensatz zu `i386` - das Alignment von `int`-Werten unbedingt auf 4-Byte-Grenzen erfolgen muss. Beschreiben Sie in Ihrer Dokumentation das Verhalten Ihrer Lösung - warum funktioniert sie, bzw. warum funktioniert sie nicht.

**Abgabe: bis spätestens Donnerstag, 04.12.2008, 14:00 Uhr**

**Hinweise zur Lösung dieser Aufgabe:**

- Die Funktionen `malloc`, `free`, `realloc` und `calloc` müssen das in den Manpages beschriebene Verhalten aufweisen. Denken Sie auch an das richtige Setzen von **errno(3)** im Fehlerfall!
- Zur Vereinfachung der Aufgabe soll nur 1 Megabyte Speicher verwaltet werden und es ist auch nicht nötig, dass freie Speicherbereiche wieder zusammengefasst werden.