# Vorbereitung

Die Dateien zum heutigen Praktikumstermin liegen in `http://fhm.hgesser.de/bs-ss2008/prakt10.tgz`. Entpacken Sie das Archiv und wechseln Sie in das neue Unterverzeichnis `prakt10`.

# 1. Speicherverwaltung: Buddy-System

Ein Betriebssystem verwende das Buddy-System für die Speicherverwaltung. Der Rechner ist mit 512 KByte RAM ausgestattet, die vollständig im Buddy-System verwaltet werden. (Dass das Betriebssystem auch Speicher braucht, soll hier vernachlässigt werden.)

Die Prozesse A, B, C und D fordern wie folgt Speicher an und geben ihn wieder frei:

1. A: 59 KByte anfordern
2. B: 27 KByte anfordern
3. C: 28 KByte anfordern
4. D: 115 KByte anfordern
5. B: Speicher freigeben
6. A: Speicher freigeben
7. D: Speicher freigeben
8. C: Speicher freigeben

a) Zeichnen Sie – ausgehend von komplett freiem und nicht in Blöcke eingeteiltem Speicher – für jeden Schritt die Speicherbelegung (jeweils nach diesem Schritt).

b) Zeichnen Sie den zugehörigen Baum im Zustand nach Schritt 4. Verwenden Sie unterschiedliche Knotenmarkierungen für komplett freie, komplett belegte und teilweise belegte Unterbäume bzw. Blattknoten.

c) Die Datei `buddy.py` im Archiv enthält den Ansatz einer Implementation des Buddy-Algorithmus' in Python: Die Funktionen `initialisieren()`, `anfordern()` und `freigeben()` übernehmen folgende Aufgaben:

- `initialisieren(n)` initialisiert einen Speicher der Größe $2^n$ KByte, der unbenutzt ist.
- `z = anfordern(x)` reserviert einen minimalen Block, wie er im Rahmen des Buddy-Systems nutzbar ist (also eine 2er-Potenz), in den `x` KByte hineinpassen, und gibt eine Block-ID zurück.
- `freigeben(z)` gibt den über die Block-ID `z` eindeutigen Block zurück.

Die Funktionen `initialisieren()` und `anfordern()` sind bereits vorhanden – überprüfen Sie, dass das Programm funktioniert, und finden Sie heraus, wie es arbeitet.

Schreiben Sie nun die Funktion `freigeben()` und entfernen Sie die Kommentarzeichen (#) vor den letzten vier Funktionsaufrufen im Programm, damit es auch die Rückgabe des Speichers durchführt.

# 2. Speicherverwaltung: Memory-Mapped Files

Memory-Mapped-Files bieten die Möglichkeit, eine Datei auf der Platte so anzusprechen, als würde es sich um einen (RAM-) Speicherbereich des Prozesses handeln – es wird ein Ausschnitt der vom Prozess ansprechbaren Speicheradressen mit einem gleich langen Bereich innerhalb der Datei identifiziert. Lese- und Schreiboperationen auf diese Speicheradressen werden also in entsprechende Operationen auf der Datei umgesetzt.

a) Die Datei `mmap-beispiel.py` enthält ein kleines Beispiel für die Verwendung einer Memory-Mapped-Datei. Testen Sie das Programm und machen Sie sich seine Funktion klar.

b) Passen Sie das Programm so an, dass es alle Kleinbuchstaben in der Datei in Großbuchstaben umwandelt – dabei soll es aber keine direkten Lese- oder Schreibzugriffe auf die Datei ausführen, sondern nur über die Variable `data` zugreifen. Tipp: `ord("A")=65`, `chr(65)="A"`.

# Lösungsblatt – Übung 10

Punkte:

| | **Name** | **Matrikelnummer** |
|---|---|---|
| **Teilnehmer 1** | | |
| **Teilnehmer 2** | | |