BuddySimulator
Abgabe von Werkstück A

Thatree Ludwig Tarek Sabbagh

Frankfurt University of Applied Sciences
Nibelungenplatz 1
60318 Frankfurt am Main

Prof. Dr. Christian Baun
Betriebssysteme und Rechnernetze

Zusammenfassung Das Ziel dieses Projektes ist, ein Programm zu entwickeln, das die Funktionsweise der „Buddy-Speicherverwaltung“ simuliert. Dazu werden IDs zur Identifikation und verschiedene Arrays zur Zuordnung der Speicherbuddies und Prozesse verwendet. Um das korrekte Teilen und Vereinen von Speicherbuddies sowie das Zuordnen von Prozesse zu garantieren, werden unterschiedliche Bedingungen und passende Algorithmen implementiert.

Heutzutage ist ein Smartphone mit 8 GB RAM Hauptspeicher keine Besonderheit mehr, jedoch wäre dies vor etwa 50 Jahren unvorstellbar gewesen. Damals war der Hauptspeicher von Großrechnern nicht einmal 1 MB groß. Um die damals begrenzte Speicherkapazität optimal nutzen zu können, wurden Speicherverwaltungssysteme entwickelt und implementiert. Dieses Projekt orientiert sich an die sogenannte „Buddy-Speicherverwaltung“, dessen Funktionsweise in einer Simulation dargestellt wird.

1 Aufgabenstellung

Für das Konzept der Buddy-Speicherverwaltung soll eine Simulation als Bash-Skript entwickelt und implementiert werden. Der Benutzer soll Prozesse zuweisen und freigeben können und zusätzlich die Größe des Hauptspeichers bestimmen können.

2 Konzept

Für die Entwicklung eines Programms ist das Konzept essentiell. Es ermöglicht die Konstruktion einer Basis und dient als Leitfaden während der Implementierung.

Dieses Konzept unterteilt sich in mehrere Kernpunkte.

2.1 Adressierung

Für die Identifizierung und Adressierung der einzelnen Speicherbuddies und Prozesse war die ursprüngliche Idee, sie als Objekte zu speichern. Da die Skript-Sprache Bash jedoch nicht auf die Objektorientierung basiert, wurde stattdessen ein ID-System verwendet. Diese IDs sind in zwei bis drei Abschnitten unterteillt (Siehe Abb. 1) und werden in Arrays gespeichert.

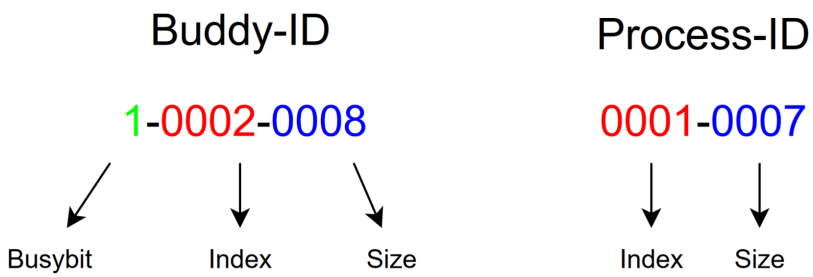

Abbildung 1. Veranschaulichung des ID-Systems

- Der erste Abschnitt prüft, ob der Buddy mit einem Prozess belegt ist. Sollte ein Buddy unbelegt sein, so zeigt der sogenannte Busybit den Wert „0“ an. Ist der Buddy allerdings belegt, geht der Wert zu „1“ über.
  Dieser Abschnitt fällt für Prozess-IDs weg.

- Im zweiten Abschnitt wird der Index der ID gespeichert. Wenn dieser Teil der ID als Index in einem Array angegeben wird, erhält man die vollständige ID als Element.

- Die Speicherkapazität der Buddies bzw. die Prozessgröße wird im letzten Abschnitt angezeigt.

2.2 Zuweisung

Wird ein Prozess einem Speicherbuddy zugewiesen, so muss diese Verbindung kenntlich gemacht werden. Das sogenannte assoziative Array ermöglicht eine solche Verbindung, indem es die Buddy-ID als Index und die Prozess-ID als Element verwendet. Der erste zugewiesene Prozess besitzt den Index-Wert „1“. Mit jeder weiteren Zuweisung steigt entsprechend auch der zugehörige Index-Wert.

Es existieren vier Arrays:

1. BuddyList[ ]: Hier werden alle Buddy-IDs gespeichert
   – z.B. `BuddyList[2]="1-0002-0008"`

2. ProcessList[ ]: Enthält alle Process-IDs
   – z.B. `ProcessList[1]="0001-0007"`

3. activeList[ ]: Beinhaltet alle aktiven Buddies und die zugewiesenen Prozesse als assoziatives Array.
   – z.B. `activeList["1-0002-0008"]="0001-0007"`

4. order[ ]: Enthält alle Prozess-Indizes und alle aktiven Buddy-IDs und ermöglicht die Ausgabe der Werte der activList[ ] in der Zuweisungsreihenfolge.
   – z.B. `order[1]="1-0002-0008"`

2.3 Passender Buddy

Um einen passenden Speicherbuddy zu finden, werden alle verfügbaren Buddies im Zusammenhang mit der Speicherkapazität überprüft. Sollte kein passender Buddy gefunden werden, wirft das Programm einen Fehler aus. Folgende Kriterien muss ein passender Buddy erfüllen:

– der Buddy darf nicht belegt sein
– der Speicherplatz des Buddys muss mindestens genau so groß sein, wie die Prozessgröße
– dennoch darf der Speicherplatz des Buddys nicht das Doppelte (oder größer) der Prozessgröße entsprechen.

Sofern mehrere passende Buddies zur Verfügung stehen, wird dieser Buddy mit dem geringsten Index-Wert ausgewählt. Sollte jedoch kein passender Buddy verfügbar sein, wird die Prozesszuteilung abgebrochen.

2.4 Teilen und Vereinen

Falls der zuzuweisende Prozess höchstens halb so groß ist, wie die Speicherkapazität des kleinsten verfügbaren Buddys, teilt sich dieser Buddy in zwei kleine Hälfen um interne Fragmentierung vorzubeugen.

Die neu entstandenen Buddies (Zwillingsbuddies) benötigen neue IDs. Diese beziehen sie aus der ursprünglichen Buddy-ID. Aus dieser ursprünglichen ID wird nur der Index (mittleres Segment) und die Größe (hinteres Segment) modifiziert. Die Zwillingsbuddies bestehen aus Buddy A, sowie Buddy B und können sich nur zu einem großen Buddy vereinen, wenn beide Zwillingsbuddies unbesetzt sind.

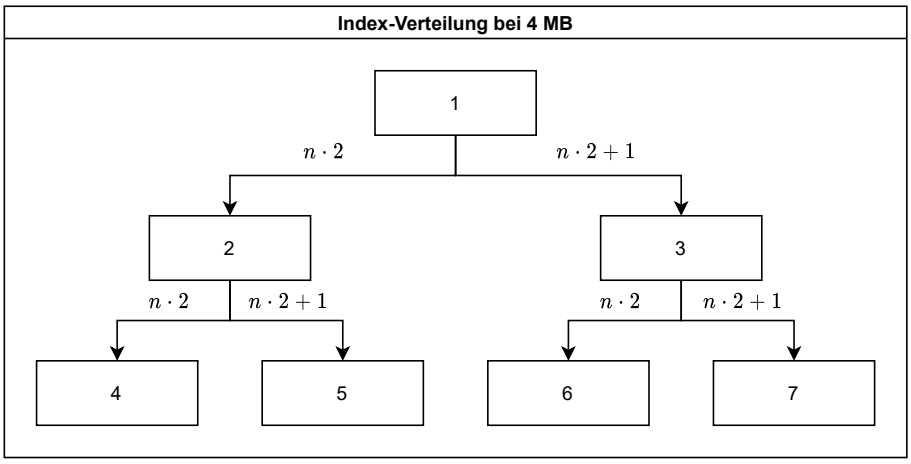

Abbildung 2. Schema für die Index-Verteilung bei 4 MB

Der Index ist essentiell für die Adressierung der Speicherbuddies. Kommt es zu Redundanzen, können falsche Informationen übergeben werden und es kommt zu einer Fehlfunktion.

Um bei der Teilung eines Buddies die Einzigartigkeit der Indizes zu wahren, müssen die Indizes nach dem Schema in der Abb. 2 verteilt sein. In diesem Fall einer „Speicherbuddy-Teilung“ erhält der Buddy A den doppelten Wert des ursprünglichen Indexes. Dieser neue Wert erhöht sich nun um „1“ und wird dem Buddy B zugeteilt. Die Zwillingsbuddies besitzen somit einen Index mit einer geraden Zahl (A) und einen mit ungerader Zahl (B).
Sollen die Zwillingsbuddies nun vereint werden, so wird der Index vom Buddy A (gerade Zahl) halbiert.

Mit dieser Methode der Index-Verteilung kommt es zu keinen Interferenzen.

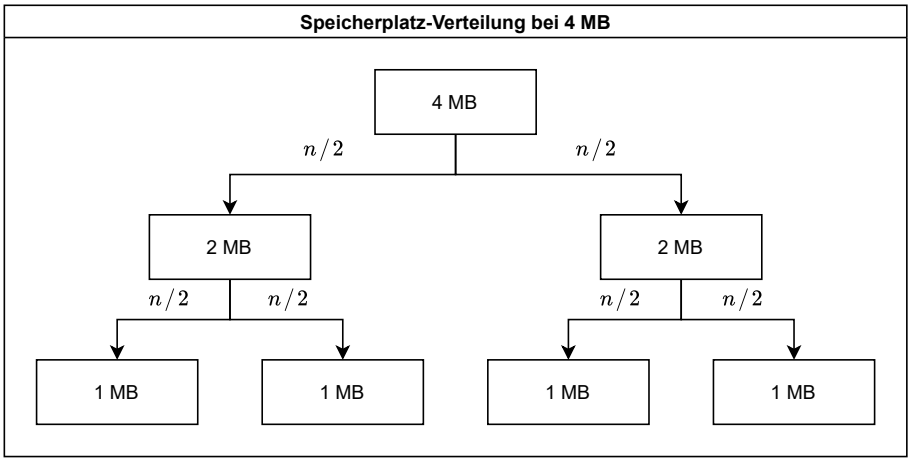

Abbildung 3. Schema für die Speicherplatz-Verteilung bei 4 MB

Wie das Schema in der Abb. 3 darstellt, teilt sich die ursprüngliche Speicherkapazität in zwei Hälften auf. Somit besitzen die neuen Buddies jeweils die Hälfte der Kapazität. Bei einer Freigabe geschieht das Gegenteil. Die zwei kleinen Buddies vereinen ihre Kapazität und vereinein sich zu einer neuen Speichereinheit mit doppeltem Speicherplatz.

Wichtig ist, dass die Gesamtkapazität einer Zweierpotenz entspricht, damit keine Teilung eines Buddys Nachkommastellen auftreten. Somit ist eine saubere Teilung der Buddies möglich (Siehe Abb. 3).

Dieses Konzept stellt die Basis für die zu implementiere Algorithmen dar.

3 Alternative Überlegungen

Zweifelsohne gibt es andere Lösungswege, die zu ähnlichen Ergebnissen führen. Statt ein assoziative Array zu verwenden, wäre ein gemeinsamer Index für die Assoziation auch möglich gewesen. Jedoch würde ein gemeinsamer Index den Überblick stören, weswegen ein assoziatives Array ihren Platz findet.

Auch hätte ein anderes ID-System verwendet werden können. Jedoch ermöglicht eine ID mit mehreren Sektoren eine schnellle Informationsaufnahme, durch die Bindestrich-Trennung.

Selbstverständlich existieren zahlreiche verschiedene Ansätze um die Simulation zu entwickeln. Nichtsdestotrotz setzte sich dieser Lösungsansatz durch, hinsichtlich der Übersicht für die Benutzer und der Zuverlässigkeit der Implementierung.

4 Implementierung

Das Programm ist in drei Bash-Skripten unterteilt, um den Überblick zu fördern und Ordnung zu bewahren.

- „BuddySimulator.bash“ ist das Haupt-Skript, welches die Simulation startet. Hier ist das Hauptmenü gespeichert
- „Library.bash“ dient als Funktionsbibliothek und beinhaltet alle Funktionen der Simulation
- „Color.bash“ bietet vorgefertigte Farben, mit der sich Strings manipulieren lassen

4.1 Anfangssequenz

Zunächst muss der Benutzer die Kapazität des Hauptspeichers bestimmen. Wie bereits erwähnt muss für die korrekte Funktionweise des Teilmechanismus eine Zweierpotenz gewählt werden.
Die Funktion `memoryCheck()` aus der Funktionsbibliothek überprüft die Eingabe. Vorerst prüft die Funktion, ob die Eingabe eine Zahl ist. Entspricht die Eingabe einer Zahl, folgen weitere Prüfungen. Besitzt die Zahl führende Nullen (z.B „0005“), so werden diese durch `parseInteger()` gelöscht. Auch die Nachkommastellen werden bei einer Dezimalzahl gelöscht. Das Ergebnis ist eine natürliche Zahl.

Um nun zu überprüfen, ob die eingegebene Zahl `n` eine Zweierpotenz ist, wird eine „Bitwise-Rechnung“durchgeführt:

```
(( n > 0 && (n & (n - 1)) == 0 ))

z.B.    n=8 (dezimal) bzw. n=1000 (binär);
    (n-1)=7 (dezimal) bzw. n=0111 (binär)
(( 8 > 0 && (1000 & 0111) == 0000 )) = 0
```

Dieser „UND-Operator“(„&“) vergleicht die Zahl `n` stellenweise im Binärsystem mit `n-1`. Sollte an keiner Stelle eine gleiche Ziffer stehen, so ist die eingegebene Zahl eine Zweierpotenz. In diesem Fall ist das Ergebnis „0“.

Bei vier ungültigen Eingaben wird das Programm abgebrochen.

Nach einer erfolgreichen Eingabe wird der Wert an `$memory` übergeben. Der Stammbuddy besitzt den Index-Wert „1“ und erhält seine Kapazität aus `$memory`

`z.B. memory=8; "0-0001-0008"`

Daraufhin wird das Hauptmenü angezeigt.

4.2 Prozesszuweisung

Will der Benutzer ein Prozess zuweisen, so muss er die Prozessgröße bestimmen. In diesem Fall gibt es eine Eingabeüberprüfung durch `processSizeCheck()`

Diese Funktion funktioniert ähnlich wie `memoryCheck()`, jedoch fällt die Zweierpotenzprüfung weg. Stattdessen darf die Zahl nicht kleiner als „1“ und nicht größer als die Gesamtkapazität sein. Auch hier formatiert `parseInteger()` die Eingabe zu einer natürlichen Zahl.

Sollte der Benutzer die Eingabe abbrechen wollen, so hat er die Möglichkeit „a“(=abbrechen) einzugeben. Daraufhin führt die Simulation zurück zum Hauptmenü.

War die Eingabe erfolgreich, so beginnt die Zuweisungssequenz, die die Abb. 4 visualisiert.

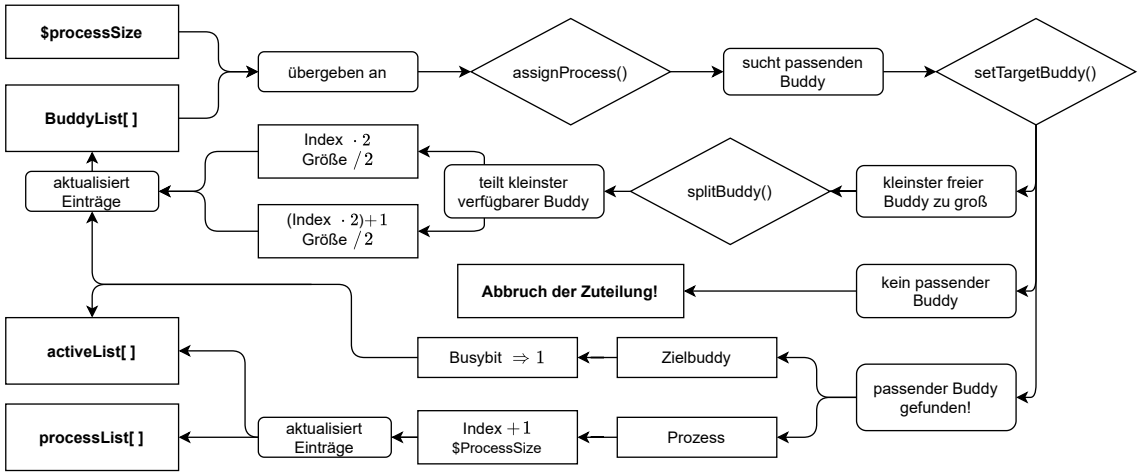

Abbildung 4. Schema für die Prozesszuweisung

Als Beispiel dient die nachfolgende Abb. 5, in der ein 2 MB großer Prozess einem leerem 8 MB großen Hauptspeicher zugewiesen wird.

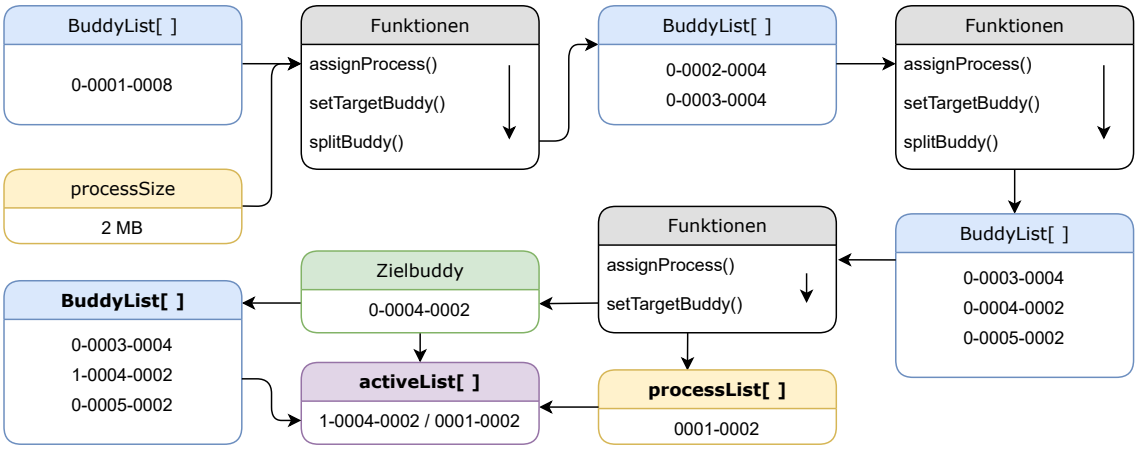

Abbildung 5. Beispiel für die Prozesszuweisung

4.3 Prozessfreigabe

Soll ein einzelner Prozess beendet werden, muss dieser ausgewählt werden. Nach der Auswahl beginnt die Freigabesequenz, die in der Abb. 6 dargestellt ist.

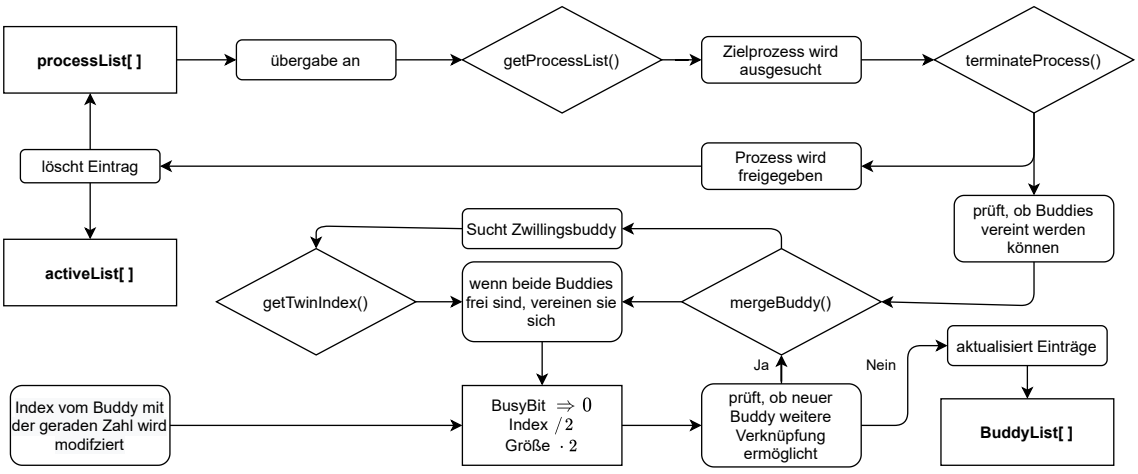

Abbildung 6. Schema für die Prozessfreigabe

Auch für diesen Fall lässt sich ein Beispiel (siehe Abb.7) darstellen. Der Zuweisungsalgorithmus und der Freigabealgorithmus besitzen ein reziprokes Verhältnis zueinander, weswegen dieses Freigabe-Beispiel die Zuweisung aus Abb. 5 anulliert.

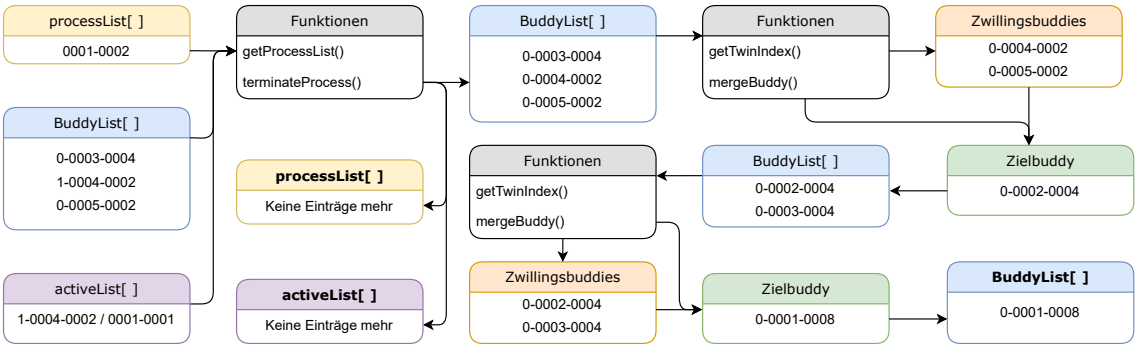

Abbildung 7. Beispiel für die Prozessfreigabe

Der Benutzer kann zudem alle Prozesse zugleich freigeben. Dies bewirkt die Auflösung der Einträge aller Arrays. Nur der Index-Wert der Prozesse fängt nicht wieder bei „1“ an, sondern wird fortgesetzt. Der Stammbuddy ist nach der Aktion wieder als einziger Buddy verfügbar.

5 Bewertung

Nach anfänglichen Problemen mit der Herangehensweise wurde das Konzept entwickelt. Das Programm erfüllt alle Anforderungen des Konzepts und bietet eine intuitve Umgebung für Benutzer. Nur die Implementierung des assoziative Arrays erwies sich als eine Hürde, die viele Fehlermeldungen verursachte. Jedoch wurden alle festgestellten Fehler behoben und die die Funktionalität der Simulation durch Eingabefilter gesichert. Insgesamt bestätigten sich jegliche Erwartungen für dieses Programm namens „BuddySimulator“.

5.1 Programmbeispiel

Abschließend präsentiert die Abb. 8 einen Zustand der Simulation. Dieses Beispiel lehnt sich an das Zuweisungsebeispiel aus Abb. 5 an.

Abbildung 8. Die Zusatzoption „Informationen anzeigen“ im Hauptmenü ermöglicht die Einsicht des momentanen Hauptspeicherzustandes.

6 Schlusswort

Die entwickelte Simulation umfasst das Konzept der „Buddy-Speicherverwaltung“. Der Einsatz eines speziellen ID-Systems ermöglicht die Umsetzung in der Skript-Sprache „Bash“ durch die Möglichkeit die Speicherbuddies und Prozesse in Form von IDs in diversen Arrays zu speichern. Mithilfe von zugeschnittenen Algorithmen und sinnvollen Bedingungen lässt sich ein genauer Ablauf des Zuteilungs- und Freigabemechanismus gewährleisten. Bei einer Zuweisung ermöglicht ein assoziatives Array die unmittelbare Verknüpfung zwischen Speicherbuddy und Prozess. Zusätzlich vermeidet die Implementierung eines Eingabefilters Fehlfunktionen im Programm.

Letzendlich bietet der „BuddySimulator“ eine benutzerfreundliche Oberfläche mit umfassender und insbesonders zuverlässigen Funktionalität.

Literatur

1. Foliensatz 2 der Vorlesung Betriebsssysteme und Rechnernetze im SS2020
2. Baun, Christian. *1. Auflage: Betriebssysteme kompakt* . Springer Vieweg. 2017
3. Alle verwendete Darstellungen entstammen der eigenen Erstellung.