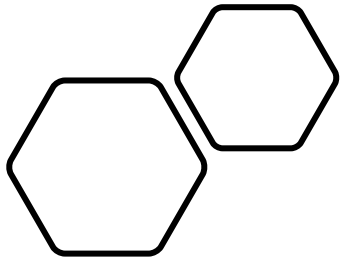
Dynamische Partitionierung



Von: Steffen Rottke und Michelle Müller

Betriebssysteme und Rechnernetze
Sommersemester 2022

29. Juli 2022

Gliederung

1 Steuerung des Simulators

2 Realisierungskonzepte

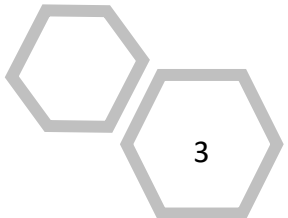
3 Visuelle und informative Darstellung der Speicherbelegung

Steuerung des Simulators

Hauptsteuerung (1/3)

```
#Benutzer wird aufgefordert, eine Gesamtspeichergroesse zu waehlen
echo "Bitte geben Sie eine Gesamtspeichergroesse an."
#Speichergroesse wird eingelesen
read -r speichergroesse
        #Nachdem die Groesse eingelesen wurde wird ueberprueft, ob die Eingabe andere Zeichen ausser Zahlen enthaelt oder nicht groesser als 0 ist
        #Wenn das der Fall ist, wird solange nach der Groesse gefragt, bis keine anderen Zeichen ausser Zahlen enthalten sind und die Eingabe groesser 0 ist
        #Als Quelle fuer die Funktionsaufrufe wird simulationssteuerungFunktionen.sh genutzt
        #Der Rueckgabewert der Funktion keineZeichenGroesserNull ist die Gesamtspeichergroesse, welche in der Variablen gewaehlteGesamtspeichergroesse gespeichert wird
        while [ -n "$(printf '%s\n' "$speichergroesse" | sed 's/[0-9]//g')" ] || [[ ! $speichergroesse -gt 0 ]]
        do
                keineZeichenNichtGroesserNull speichergroesse
                zeichen speichergroesse
                read -r speichergroesse
        done
        gewaehlteGesamtspeichergroesse=$(keineZeichenGroesserNull)
echo "Ihre gewaehlte Speichergroesse ist: $gewaehlteGesamtspeichergroesse"
```

- Abfrage nach der Gesamtspeichergröße
- Überprüfung durch eine while-Schleife
- Rückgabewert der Funktion *keineZeichenGroesserNull*
- Wert wird in der Variable *gewaehlteGesamtspeichergroesse* gespeichert

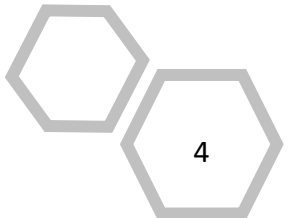


Steuerung des Simulators

Hauptsteuerung (2/3)

```
#Benutzer wird aufgefordert, das Realisierungskonzept auszuwaehlen, es gibt die Wahl zwischen den Shortcuts f,b,n,w,r und l
echo "Bitte geben Sie ein, welches Realisierungskonzept genutzt werden soll. Es stehen folgende Realisierungskonzepte zur Verfuegung:
-> f (First Fit)
-> b (Best Fit)
-> n (Next Fit)
-> w (Worst Fit)
-> r (Random)
-> l (Last Fit)"
#Das ausgewaehlte Konzept wird eingelesen
read -r konzept
        #Als Quelle wird die Datei simulationssteuerungFunktionen.sh genutzt
        #Es wird ueberprueft, ob die Eingabe einem der Realisierungskonzepte entspricht oder ob erneut nach einer Eingabe gefragt wird
        while ! { [ "$konzept" = "f" ] || [ "$konzept" = "b" ] || [ "$konzept" = "n" ] || [ "$konzept" = "w" ] || [ "$konzept" = "r" ] || [ "$konzept" = "l" ]; }
        do
                ungueltigesRealisierungskonzept konzept
                read -r konzept
        done
```

- Abfrage nach dem Realisierungskonzept
- Überprüfung durch eine while-Schleife



Steuerung des Simulators

Hauptsteuerung (3/3)

```
case "$konzept" in
"f")
        echo "Ihr gewaehltes Konzept ist: First Fit."
        #Die Funktion abfrageAktion wird von der Quelle simulationssteuerungFunktionen.sh aufgerufen
        abfrageAktion
        #Der Befehl wird eingelesen
        #Es wird die Funktion ablaufRealisierungskonzept aufgerufen und der eingelesene Befehl uebergeben
        #Nachdem der jeweilige Befehl ausgewaehlt wurde (alle moeglich ausser q), wird die Funktion insertFirstFit aufgerufen, wenn der Befehl ein c ist
        #Der Funktion werden dabei drei Werte uebergeben
        #Nachdem die Funktion ausgefuehrt wurde, wird erneut nach einer Eingabe gefragt
        read -r befehl
                while [ "$befehl" != "q" ]
                do
                        ablaufRealisierungskonzept befehl
                        if [ "$befehl" = "c" ]; then
                        insertFirstFit $gewaehlteGesamtspeichergroesse "$createPartitionsName" $createPartitionsGroesse
                        fi
                        #Die Speicherbelegung wird nun ausgegeben
                        echo ""
                        echo "Die Speicherbelegung sieht wie folgt aus: "
                        #Visuelle Ausgabe des Speichers und der Informationen zur Speicherbelegung durch den Funktionsaufruf von visualout aus der Datei visualout.sh
                        visualout
                        abfrageAktion
                        read -r befehl
                done
        #Wenn der Befehl quit gewaehlt wurde, wird nun ausgegeben, dass der Simulator beendet wurde und im Hintergrund wird die Speicherbelegung zurueckgesetzt
        echo "Der Simulator wurde beendet."
        ;;
```

Realisierungskonzept First Fit

- Funktionsaufruf für die Abfrage der auszuführenden Aktion
- Eine while-Schleife zum Überprüfen auf *q*
- Funktionsaufruf von *ablaufRealisierungskonzept*
- Eventuell Aufruf von *insertFirstFit*
- Ausgabe der Speicherbelegung
- Erneute Abfrage nach der auszuführenden Aktion
- Je nach Eingabe des Befehls wird der Simulator beendet

Steuerung des Simulators

Simulationssteuerung (1/2)

```
case  "$befehl" in
"c")
        #Wenn der Befehl create entspricht, wird nun die Abfrage nach dem Namen und anschliessend nach der Groesse gestartet
        #Die Variablen createPartitionsName und createPartitionsGroesse speichern den Namen bzw. die jeweilige Groesse bis zu dem eingegebenen c
        echo 'Bitte geben Sie einen Namen für die Partition ein und bestaetigen Sie mit: " c".'
                read -r name
                        #Nachdem der Name eingelesen wurde wird ueberprueft, ob die Einagbe mit einem c bestaetigt wurde oder nicht
                        #Wenn nicht, wird solange nach dem Namen gefragt, bis mit c bestaetigt wurde
                        #Die Funktionen zur Ueberpruefung befinden sich in simulationssteuerungFunktionen.sh
                                while [[ ! "$name" =~ " c" ]]
                                do
                                        frageNachNameNichtBestaetigt name
                                        read -r name
                                done
                                createPartitionsName=$(frageNachNameBestaetigt)
                                #Die Funktion pruefenAufGleichenNamen ueberprueft, dass eine erstellte Partition nicht den gleichen Namen hat wie eine bereits erstellte
                                pruefenAufGleichenNamen $(frageNachNameBestaetigt)
        echo 'Bitte geben Sie eine Groesse für die Partition ein und bestaetigen Sie mit: " c".'
                        read -r groesse
                        #Nachdem die Groesse eingelesen wurde wird ueberprueft, ob die Eingabe mit einem c bestaetigt wurde oder nicht und ob es sich um eine Zahl groesser$
                        #Wenn nicht, wird solange nach der Groesse gefragt, bis mit c bestaetigt wurde und die Eingabe nur Zahlen groesser 0 enthaelt
                        #Die Funktionen zur Ueberpruefung befinden sich in simulationssteuerungFunktionen.sh
                                while [ -n "$(printf '%s\n' "${groesse/ c*/}" | sed 's/[0-9]//g')" ] || [[ ! "$groesse" =~ " c" ]] || [[ ! ${groesse/ c*/} -gt 0 ]]
                                do
                                        zeichenBestaetigt groesse
                                        zeichenNichtBestaetigt groesse
                                        keineZeichenBestaetigtNichtGroesserNull groesse
                                        keineZeichenBestaetigtGroesserNull groesse
                                        keineZeichenNichtBestaetigtNichtGroesserNull groesse
                                        keineZeichenNichtBestaetigtGroesserNull groesse
                                        read -r groesse
                                done
                        createPartitionsGroesse=$(keineZeichenBestaetigtGroesserNull)
        echo "Der eingegebene Name für die Partition ist: $createPartitionsName und die gewaehlte Groesse ist: $createPartitionsGroesse."
        ;;
```

Erstellen einer neuen Partition

- Abfrage nach dem Partitionsnamen und Überprüfung des Namens mit einer while-Schleife
- Der Name wird in der Variable *createPartitionsName* gespeichert und es wird auf doppelte Namensvergabe überprüft
- Abfrage nach Partitionsgröße und Überprüfung der Größe mit einer while-Schleife
- Die Größe wird in der Variable *createPartitionsGroesse* gespeichert

Steuerung des Simulators

Simulationssteuerung (2/2)

```
"n")
        #Die Datei simulationssteuerungFunktionen wird aufgerufen, um auf die Funktion speicherZuruecksetzen zuzugreifen, die die derzeitigen Namen, die Groessen und den S$
        speicherZuruecksetzen save_name save_groesse save_speicher
        ;;
```

Zurücksetzen der Speicherbelegung

- Funktionsaufruf von *speicherZuruecksetzen* und die Übergabe der drei Arrays mit den Namen, den Größen und dem Speicher

Steuerung des Simulators

Funktionen für die Haupt- und Simulationssteuerung

```
#Funktion fuer die Partitionen zum Ueberpruefen, dass keine Zeichen in der Eingabe vorhanden sind, die Eingabe mit c bestaetigt wurde und die Eingabe groesser 0 ist
keineZeichenBestaetigtGroesserNull () {
        if [ ! -n "$(printf '%s\n' "${groesse/ c*/}" | sed 's/[0-9]//g')" ] && [[ "$groesse" =~ " c" ]] && [[ ${groesse/ c*/} -gt 0 ]]; then
                echo ${groesse/ c*/}
        fi
}
```

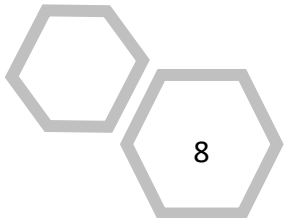
Funktion *keineZeichenBestaetigtGroesserNull*

- Wird ausgeführt, wenn keine anderen Zeichen außer Zahlen in der Eingabe vorhanden sind, die Eingabe mit einem *c* bestätigt wurde und die Eingabe größer als null ist
- Die Funktion gibt die eingegebene Größe ohne das Leerzeichen und das *c* zurück

```
ablaufRealisierungskonzept () {
        source ./simulationssteuerung.sh
        if [ ! "$befehl" = "q" ]; then
                simulationssteuerung befehl
        fi
}
```

Funktion *ablaufRealisierungskonzept*

- Wird ausgeführt, wenn kein *q* eingegeben wurde
- Die Datei *simulationssteuerung.sh* ist die Quelle für den Funktionsaufruf *simulationssteuerung*
- Die Funktion gibt den zuvor eingegebenen Befehl an die Funktion weiter



Realisierungskonzepte

Ablauf der Datenverwaltung

- FirstFit
  - Befüllt den Speicher startend von vorne
  - Der erste Speicherblock der groß genug ist wird belegt
- LastFit
  - Befüllt den Speicher startend von hinten
  - Der erste Speicherblock der groß genug ist wird belegt
- NextFit
  - Befüllt den Speicher startend von vorne
  - Startet an der Stelle wo zuletzt gespeichert wurde
- RandomFit
  - Befüllt den Speicher in einer zufälligen Reihenfolge
  - Eine Zufallszahl aus einem Pool an verfügbaren Speicherblöcken wird als Speicherort gewählt
- BestFit
  - Befüllt den Speicher an der Stelle, wo es am wenigsten Fragmentierung gibt
- WorstFit
  - Befüllt den Speicher startend an der Stelle, wo es am meisten Fragmentierung gibt
  - Zusätzlich: Platziert sich in der Mitte vom vorhandenen Speicherplatz

Realisierungskonzepte

Probleme der Konzepte

- FirstFit
  - Es könnte einen effizienteren Speicherort geben als den ersten der frei ist und passt
- LastFit
  - Es könnte einen effizienteren Speicherort geben als den ersten der frei ist und passt (von hinten)
- NextFit
  - Sobald nach mindestens einem Create etwas gelöscht wird, wird dieser Speicherort erstmals ignoriert, sofern man nicht am Speicherende ist
- RandomFit
  - Sehr geringer Einfluss auf den Speicherort
- BestFit
  - Erhöhter Rechenaufwand
- WorstFit
  - Erhöhter Rechenaufwand
  - Massive Fragmentierung

Realisierungskonzepte

Mögliche Einsätze der Verfahren

- FirstFit
  - In Bereichen, in denen Daten so schnell wie möglich abgespeichert werden sollen
  - Effizienz: Mittel
- LastFit
  - In Bereichen, in denen Daten so schnell wie möglich abgespeichert werden sollen
  - Effizienz: Mittel
- NextFit
  - In Bereichen, in denen ältere Dateien nicht oft gelöscht oder geändert werden
  - Effizienz: Mittel (-)
- RandomFit
  - In Bereichen, in denen Zugriffszeiten ähnlich sein sollen
  - Effizienz: Mittel
- BestFit
  - In Bereichen, in denen Speichereffizienz eine größere Rolle spielt als die Speicherdauer
  - Effizienz: Super
- WorstFit
  - Niemals.
  - Effizienz: Schlecht (-)

Realisierungskonzepte

BestFit

FirstFit

LastFit

Visuelle und informative Darstellung der Speicherbelegung

Datenbeschaffung

```
#Beschaffung der Daten
vo_name="${save_name[@]}"
vo_speicher="${save_speicher[@]}"
vo_groesse="${save_groesse[@]}"
vo_max=$gewaehlteGesamtspeichergroesse
```

Visuelle und informative Darstellung der Speicherbelegung

Datenaufbereitung

```
for (( i=0; i<$save_speicherplatz; i++ ))
do

    #Ausgabevorbereitung Prozessname
    if [[ -z "${save_name[$i]}" ]]; then
        nameout[$i]="X"
    else
        nameout[$i]=${save_name[$i]}
    fi

    #Ausgabevorbereitung Prozessgroesse
    if [[ -z "${save_groesse[$i]}" ]]; then
        groesseout[$i]="X"
    else
        groesseout[$i]=${save_groesse[$i]}
    fi

    #Ausgabevorbereitung Speicher und Berechnung vom groessten und kleinsten Block
    if [[ -z "${save_speicher[$i]}" ]]; then
        speicherout[$i]="X"
    else
        speicherout[$i]=${save_speicher[$i]}
        if [[ ${save_speicher[$i]} -gt $vo_groessterblock ]]; then
            vo_groessterblock=${save_speicher[$i]}
        fi
        if [[ ${save_speicher[$i]} -lt $vo_kleinsterblock ]]; then
            vo_kleinsterblock=${save_speicher[$i]}
        fi
    fi
done

if [[ $vo_kleinsterblock -eq $(($save_speicherplatz+1)) ]]; then
    unset vo_kleinsterblock
fi
```

Visuelle und informative Darstellung der Speicherbelegung

Ausgabe der aufbereiteten Daten

```
#Visualisierung der Prozesse
echo ${speicherout[*]}
echo "++++++++++"
    echo "----------"

    echo "**********"
    echo "Gespeicherte Namen: ${nameout[*]}"
    echo "**********"

    echo "----------"
    echo "Gespeicherte Größen: ${groesseout[*]}"
    echo "----------"

    echo ""
    echo "++++++++++"
    echo "Belegung: $vo_belegterblock von $vo_max"
    echo "Anzahl belegte Blöcke: $vo_belegterblock"
    echo "Anzahl freie Blöcke: $(($vo_max-$vo_belegterblock))"
    echo "Größter Block: $vo_groessterblock"
    echo "Kleinster Block: $vo_kleinsterblock"
    echo "++++++++++"
```

Visuelle und informative Darstellung der Speicherbelegung

Visuelle Beispielausgabe

Vielen Dank für Ihre Aufmerksamkeit!

Gibt es Fragen?