

Lösung von Übungsblatt 6

Aufgabe 1 (Interprozesskommunikation)

1. Beschreiben Sie was ein kritischer Abschnitt ist.

Mehrere Prozesse greifen lesend und schreibend auf gemeinsame Daten zu.

2. Beschreiben Sie was eine Race Condition ist.

Eine unbeabsichtigten Wettlaufsituation zweier Prozesse, die auf die gleiche Speicherstelle schreibend zugreifen wollen.

3. Erklären Sie warum Race Conditions schwierig zu lokalisieren und zu beheben sind.

Das Ergebnis eines Prozesses hängt von der Reihenfolge oder dem zeitlichen Ablauf anderer Ereignisse ab. Bei jedem Testdurchlauf können die Symptome komplett verschieden sein oder verschwinden.

4. Beschreiben Sie wie Race Conditions vermieden werden.

Durch das Konzept der Semaphore.

5. Wenn zwei Prozesse auf gemeinsame Ressourcen (z. B. Daten) zugreifen, bezeichnet man ihre Beziehung als...

- | | | |
|---|--|---------------------------------------|
| <input type="checkbox"/> Allokation | <input type="checkbox"/> Kommunikation | <input type="checkbox"/> Alles davon |
| <input checked="" type="checkbox"/> Kooperation | <input type="checkbox"/> virtuell | <input type="checkbox"/> Nichts davon |

(Hinweis: Eine einzige Antwort ist korrekt.)

6. Wenn ein Prozess eine Kopie seiner Daten zu einem zweiten Prozess sende, bezeichnet man ihre Beziehung als...

- | | | |
|--------------------------------------|---|---------------------------------------|
| <input type="checkbox"/> Allokation | <input checked="" type="checkbox"/> Kommunikation | <input type="checkbox"/> Alles davon |
| <input type="checkbox"/> Kooperation | <input type="checkbox"/> virtuell | <input type="checkbox"/> Nichts davon |

(Hinweis: Eine einzige Antwort ist korrekt.)

7. Kreuze das Konzept an, das elementar für die Antworten aus den beiden vorherigen Teilaufgaben ist.

- | | | |
|---|---------------------------------------|---|
| <input type="checkbox"/> Orchestrierung | <input type="checkbox"/> Highlighting | <input type="checkbox"/> Parallelisierung |
| <input type="checkbox"/> Serialisierung | <input type="checkbox"/> Bypassing | <input checked="" type="checkbox"/> Synchronisation |

(Hinweis: Eine einzige Antwort ist korrekt.)

Aufgabe 2 (Synchronisation)

1. Beschreiben Sie den Vorteil von Signalisieren und Warten gegenüber aktivem Warten (Warteschleife).

Bei aktivem Warten wird Rechenzeit der CPU verschwendet, weil diese immer wieder vom wartenden Prozess belegt wird. Bei Signalisieren und Warten wird die CPU entlastet, weil der wartende Prozess blockiert und zu einem späteren Zeitpunkt deblockiert wird.

2. Beschreiben Sie was eine Barriere ist.

Eine Barriere synchronisiert die beteiligten Prozesse an einer Stelle.

3. Nennen Sie die beiden Probleme, die durch Blockieren entstehen können.

Verhungern (Starving) und Verklemmung (Deadlock).

4. Beschreiben Sie den Unterschied zwischen Signalisieren und Blockieren.

Signalisieren legt die Ausführungsreihenfolge der kritische Abschnitte der Prozesse fest.

Blockieren sichert kritische Abschnitte. Die Reihenfolge, in der die Prozesse ihre kritische Abschnitte abarbeiten, ist nicht festgelegt. Es wird nur sichergestellt, dass es keine Überlappung in der Ausführung der kritischen Abschnitte gibt.

5. Markieren Sie die vier Bedingungen, die gleichzeitig erfüllt sein müssen, damit ein Deadlock entstehen kann.

- | | |
|--|---|
| <input type="checkbox"/> Rekursive Funktionsaufrufe | <input checked="" type="checkbox"/> Anforderung weiterer Betriebsmittel |
| <input checked="" type="checkbox"/> Wechselseitiger Ausschluss | <input type="checkbox"/> > 128 Prozesse im Zustand blockiert |
| <input type="checkbox"/> Häufige Funktionsaufrufe | <input type="checkbox"/> Iterative Programmierung |
| <input type="checkbox"/> Geschachtelte for -Schleifen | <input checked="" type="checkbox"/> Zyklische Wartebedingung |
| <input checked="" type="checkbox"/> Ununterbrechbarkeit | <input type="checkbox"/> Warteschlangen |

6. Führen Sie eine Deadlock-Erkennung mit Matrizen durch und prüfen Sie ob es zum Deadlock kommt.

$$\text{Ressourcenvektor} = (8 \ 6 \ 7 \ 5)$$

$$\text{Belegungsmatrix} = \begin{bmatrix} 2 & 1 & 0 & 0 \\ 3 & 1 & 0 & 4 \\ 0 & 2 & 1 & 1 \end{bmatrix} \quad \text{Anforderungsmatrix} = \begin{bmatrix} 3 & 2 & 4 & 5 \\ 1 & 1 & 2 & 0 \\ 4 & 3 & 5 & 4 \end{bmatrix}$$

Aus dem Ressourcenvektor und der Belegungsmatrix ergibt sich der Ressourcenrestvektor.

$$\text{Ressourcenrestvektor} = (3 \ 2 \ 6 \ 0)$$

Nur Prozess 2 kann bei diesem Ressourcenrestvektor laufen. Folgender Ressourcenrestvektor ergibt sich, wenn Prozess 2 beendet ist und seine Ressourcen freigegeben hat.

$$\text{Ressourcenrestvektor} = (6 \ 3 \ 6 \ 4)$$

Nur Prozess 3 kann bei diesem Ressourcenrestvektor laufen. Folgender Ressourcenrestvektor ergibt sich, wenn Prozess 3 beendet ist und seine Ressourcen freigegeben hat.

$$\text{Ressourcenrestvektor} = (6 \ 5 \ 7 \ 5)$$

Nun kann Prozess 1 laufen.

Es kommt nicht zum Deadlock.

7. Ein Nachteil der Deadlock-Erkennung mit Betriebsmittel-Graphen ist, dass sie nicht verwendet werden kann. . .
- weil das Verfahren nur maximal drei Prozesse darstellen kann.
 - wenn ein Prozess verhungert ist.
 - weil sie die Ressourcen nur zu einem einzigen Zeitpunkt darstellen kann.
 - wenn es mehrere Kopien (Instanzen) einer Ressource gibt.

Aufgabe 3 (Kommunikation von Prozessen)

1. Beschreiben Sie was bei Interprozesskommunikation über gemeinsame Speichersegmente (Shared Memory) zu beachten ist.

Die Prozesse müssen die Zugriffe selbst koordinieren und sicherstellen, dass ihre Speicherzugriffe sich gegenseitig ausschließen. Der Sender-Prozess darf nichts aus dem gemeinsamen Speicher lesen, bevor der Sender-Prozess fertig geschrieben hat. Ist die Koordinierung der Zugriffe nicht sorgfältig \implies Inkonsistenzen.

2. Beschreiben Sie die Aufgabe die Shared Memory Tabelle im Linux-Kernel.

Unter Linux/UNIX speichert eine Shared Memory Tabelle mit Informationen über die existierenden gemeinsamen Speichersegmente. Zu diesen Informationen gehören: Anfangsadresse im Speicher, Größe, Besitzer (Benutzername und Gruppe) und Zugriffsrechte.

3. Kreuzen Sie an, welche Auswirkungen ein Neustart (Reboot) des Betriebssystems auf die bestehenden gemeinsamen Speichersegmente (Shared Memory) hat.

(Nur eine Antwort ist korrekt!)

Die gemeinsamen Speichersegmente werden beim Neustart erneut angelegt und die Inhalte werden wieder hergestellt.

Die gemeinsamen Speichersegmente werden beim Neustart erneut angelegt, bleiben aber leer. Nur die Inhalte sind also verloren.

Die gemeinsamen Speichersegmente und deren Inhalte sind verloren.

Nur die gemeinsamen Speichersegmente sind verloren. Die Inhalte speichert das Betriebssystem in temporären Dateien im Ordner `\tmp`.

4. Markieren Sie das Funktionsprinzip von Nachrichtenwarteschlangen (Message Queues).

(Nur eine Antwort ist korrekt!)

Round Robin

LIFO

FIFO

SJF

LJF

5. Geben Sie an, wie viele Prozesse über eine Pipe miteinander kommunizieren können. *Pipes können immer nur zwischen 2 Prozessen tätig sein.*

6. Beschreiben Sie den Effekt, wenn ein Prozess in eine volle Pipe schreiben will.

Der in die Pipe schreibende Prozess wird blockiert.

7. Beschreiben Sie den Effekt, wenn ein Prozess aus einer leeren Pipe lesen will.

Der aus der Pipe lesende Prozess wird blockiert.

8. Nennen Sie die beiden Arten von Pipes.

Anonyme Pipes und benannte Pipes.

9. Nennen Sie die beiden Arten von Sockets.

Verbindungslose Sockets (bzw. Datagram Sockets) und verbindungsorientierte Sockets (bzw. Stream Sockets).

10. Kommunikation via Pipes funktioniert...

(Nur eine Antwort ist korrekt!)

speicherbasiert

datenstrombasiert

objektbasiert

nachrichtenbasiert

11. Kommunikation via Nachrichtenwarteschlangen funktioniert. . .
(Nur eine Antwort ist korrekt!)
- | | |
|--|--|
| <input type="checkbox"/> speicherbasiert | <input type="checkbox"/> datenstrombasiert |
| <input type="checkbox"/> objektbasiert | <input checked="" type="checkbox"/> nachrichtenbasiert |
12. Kommunikation via gemeinsamen Speichersegmenten funktioniert. . .
(Nur eine Antwort ist korrekt!)
- | | |
|---|---|
| <input checked="" type="checkbox"/> speicherbasiert | <input type="checkbox"/> datenstrombasiert |
| <input type="checkbox"/> objektbasiert | <input type="checkbox"/> nachrichtenbasiert |
13. Kommunikation via Sockets funktioniert. . .
(Nur eine Antwort ist korrekt!)
- | | |
|--|--|
| <input type="checkbox"/> speicherbasiert | <input type="checkbox"/> datenstrombasiert |
| <input type="checkbox"/> objektbasiert | <input checked="" type="checkbox"/> nachrichtenbasiert |
14. Geben Sie an, welche Form der Interprozesskommunikation bidirektional funktioniert.
- | | |
|---|--|
| <input checked="" type="checkbox"/> Gemeinsame Speichersegmente | <input type="checkbox"/> Nachrichtenwarteschlangen |
| <input type="checkbox"/> Anonyme Pipes | <input type="checkbox"/> Benannte Pipes |
| <input checked="" type="checkbox"/> Sockets | |
15. Geben Sie an, welche Form der Interprozesskommunikation nur zwischen Prozessen funktioniert die eng verwandt sind.
- | | |
|--|--|
| <input type="checkbox"/> Gemeinsame Speichersegmente | <input type="checkbox"/> Nachrichtenwarteschlangen |
| <input checked="" type="checkbox"/> Anonyme Pipes | <input type="checkbox"/> Benannte Pipes |
| <input type="checkbox"/> Sockets | |
16. Geben Sie an, welche Form der Interprozesskommunikation über Rechnergrenzen funktioniert.
- | | |
|--|--|
| <input type="checkbox"/> Gemeinsame Speichersegmente | <input type="checkbox"/> Nachrichtenwarteschlangen |
| <input type="checkbox"/> Anonyme Pipes | <input type="checkbox"/> Benannte Pipes |
| <input checked="" type="checkbox"/> Sockets | |
17. Geben Sie an, bei welchen Formen der Interprozesskommunikation die Daten auch ohne gebundenen Prozess erhalten bleiben.
- | | |
|---|---|
| <input checked="" type="checkbox"/> Gemeinsame Speichersegmente | <input checked="" type="checkbox"/> Nachrichtenwarteschlangen |
| <input type="checkbox"/> Anonyme Pipes | <input type="checkbox"/> Benannte Pipes |
| <input type="checkbox"/> Sockets | |
18. Geben Sie an, bei welcher Form der Interprozesskommunikation das Betriebssystem die Synchronisierung nicht garantiert.
- | | |
|---|--|
| <input checked="" type="checkbox"/> Gemeinsame Speichersegmente | <input type="checkbox"/> Nachrichtenwarteschlangen |
| <input type="checkbox"/> Anonyme Pipes | <input type="checkbox"/> Benannte Pipes |
| <input type="checkbox"/> Sockets | |

Aufgabe 4 (Kooperation von Prozessen)

1. Beschreiben Sie was eine Semaphore ist und beschreiben Sie ihren Einsatzzweck.

Ein Semaphore ist eine Zählersperre.

2. Nennen Sie die beiden Operationen, die Semaphore verwenden.
Gesucht sind die Bezeichnungen und eine (kurze) Beschreibung der Funktionsweise.

Die Zugriffsoperationen $P(S)$ versucht den Wert der Zählvariable S zu verringern.

Die Zugriffsoperationen $V(S)$ erhöht den Wert der Zählvariable S .

3. Beschreiben Sie den Unterschied zwischen Semaphore und Blockieren (Sperren und Freigeben).

Im Gegensatz zu Semaphore kann beim Blockieren (Sperren und Freigeben) immer nur ein Prozess den kritischen Abschnitt betreten.

4. Nennen Sie das Linux/UNIX-Kommando, das Informationen zu bestehenden gemeinsamen Speichersegmenten, Nachrichtenwarteschlangen und Semaphore liefert.

`ipcs`

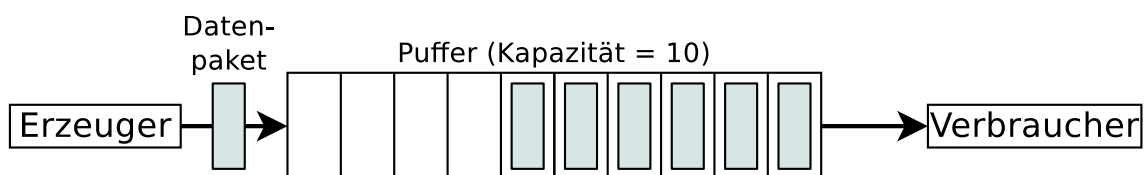
5. Nennen Sie das Linux/UNIX-Kommando, das es ermöglicht bestehende gemeinsame Speichersegmente, Nachrichtenwarteschlangen und Semaphore zu löschen.

`ipcrm`

Aufgabe 5 (Erzeuger/Verbraucher-Szenario)

Ein Erzeuger soll Daten an einen Verbraucher schicken. Ein endlicher Zwischenspeicher (Puffer) soll die Wartezeiten des Verbrauchers minimieren. Daten werden vom Erzeuger in den Puffer gelegt und vom Verbraucher aus diesem entfernt. Gegensei-

tiger Ausschluss ist nötig, um Inkonsistenzen zu vermeiden. Ist der Puffer voll, muss der Erzeuger blockieren. Ist der Puffer leer, muss der Verbraucher blockieren.



Synchronisieren Sie die beiden Prozesse, indem Sie die nötigen Semaphoren erzeugen, diese mit Startwerten versehen und Semaphor-Operationen einfügen.

```
typedef int semaphore;           // Semaphore sind von Typ Integer
semaphore voll = 0;             // zählt die belegten Plätze im Puffer
semaphore leer = 10;           // zählt die freien Plätze im Puffer
semaphore mutex = 1;           // steuert Zugriff auf kritische Bereiche

void erzeuge (void) {
    int daten;

    while (TRUE) {              // Endlosschleife
        erzeugeDatenpaket(daten); // erzeuge Datenpaket
        P(leer);                 // Zähler "leere Plätze" erniedrigen
        P(mutex);               // in kritischen Bereich eintreten
        einfüegenDatenpaket(daten); // Datenpaket in Puffer schreiben
        V(mutex);               // kritischen Bereich verlassen
        V(voll);                 // Zähler für volle Plätze erhöhen
    }
}

void verbrauche (void) {
    int daten;

    while (TRUE) {              // Endlosschleife
        P(voll);                 // Zähler "volle Plätze" erniedrigen
        P(mutex);               // in kritischen Bereich eintreten
        entferneDatenpaket(daten); // Datenpaket aus dem Puffer holen
        V(mutex);               // kritischen Bereich verlassen
        V(leer);                 // Zähler für leere Plätze erhöhen
        verbraucheDatenpaket(daten); // Datenpaket nutzen
    }
}
```

Aufgabe 6 (Semaphoren)

In einer Lagerhalle werden ständig Pakete von einem Lieferanten angeliefert und von zwei Auslieferern abgeholt. Der Lieferant und die Auslieferer müssen dafür ein Tor durchfahren. Das Tor kann immer nur von einer Person durchfahren werden. Der Lieferant bringt mit jeder Lieferung 3 Pakete zum Wareneingang. An der Ausgabe holt ein Auslieferer jeweils 2 Pakete ab, der andere Auslieferer 1 Paket.

```
sema tor      = 1
sema ausgabe = 1
sema frei    = 10
sema belegt  = 0
```

Lieferant	Auslieferer_X	Auslieferer_Y
<pre>{ while (TRUE) { P(tor); <Tor durchfahren>; V(tor); <Wareneingang betreten>; P(frei); P(frei); P(frei); <3 Pakete entladen>; V(belegt); V(belegt); V(belegt); <Wareneingang verlassen>; P(tor); <Tor durchfahren>; V(tor); } }</pre>	<pre>{ while (TRUE) { P(tor); <Tor durchfahren>; V(tor); P(ausgabe); <Warenausgabe betreten>; P(belegt); P(belegt); <2 Pakete aufladen>; V(frei); V(frei); <Warenausgabe verlassen>; V(ausgabe); P(tor); <Tor durchfahren>; V(tor); } }</pre>	<pre>{ while (TRUE) { P(tor); <Tor durchfahren>; V(tor); P(ausgabe); <Warenausgabe betreten>; P(belegt); <1 Paket aufladen>; V(frei); <Warenausgabe verlassen>; V(ausgabe); P(tor); <Tor durchfahren>; V(tor); } }</pre>

Es existiert genau ein Prozess `Lieferant`, ein Prozess `Auslieferer_X` und ein Prozess `Auslieferer_Y`.

Synchronisieren Sie die beiden Prozesse, indem Sie die nötigen Semaphoren erzeugen, diese mit Startwerten versehen und Semaphor-Operationen einfügen.

Folgende Bedingungen müssen erfüllt sein:

- Es darf immer nur ein Prozess das Tor durchfahren.
- Es darf immer nur einer der beiden Auslieferer die Warenausgabe betreten.
- Es soll möglich sein, dass der Lieferant und ein Auslieferer gleichzeitig Waren entladen bzw. aufladen.
- Die Lagerhalle kann maximal 10 Pakete aufnehmen.
- Es dürfen keine Verklemmungen auftreten.
- Zu Beginn sind keine Pakete in der Lagerhalle vorrätig und das Tor, der Wareneingang und die Warenausgabe sind frei.

Quelle: TU-München, Übungen zur Einführung in die Informatik III, WS01/02

Aufgabe 7 (Shell-Skripte, Datenkompression)

1. Schreiben Sie ein Shell-Skript, das eine Datei `testdaten.txt` erzeugt.

- Die Datei soll mit Nullen gefüllt werden.
- Die Nullen liefert die virtuelle Gerätedatei `/dev/zero`.
(Beispiel: `dd if=/dev/zero of=/pfad/zur/datei bs=512 count=1`)
- Die Dateigröße soll mindestens 128 und maximal 512 kB sein.
- Wie groß die Datei wird, soll mit `RANDOM` zufällig festgelegt werden.

```
1 #!/bin/bash
2 #
3 # Skript: testdaten_erzeugen.bat
4 #
5 # falls Ordner nicht vorhanden, Ordner erzeugen
6
7 VERZEICHNIS=/tmp/testdaten
8 DATEINAME=testdaten.txt
9
10 if [ ! -d $VERZEICHNIS ] ; then
11     if mkdir $VERZEICHNIS ; then
12         echo "Ein Verzeichnis für Testdaten wurde erstellt."
13     else
14         echo "Es konnte kein Verzeichnis erstellt werden."
15     fi
16 else
17     echo "Ein Verzeichnis für Testdaten existiert schon."
18     exit 1
19 fi
20
21 if touch `echo "$VERZEICHNIS/$DATEINAME"` ; then
22     # Zufallszahl zwischen 128 und 512 erstellen
23     ZUFALLSZAHL=`awk -vmin=128 -vmax=512 'BEGIN{srand(); print
24         int(min+rand()*(max-min+1))}'`
25     # Die Datei mit Nullen füllen
26     `dd if=/dev/zero of=$VERZEICHNIS/$DATEINAME bs=$ZUFALLSZAHL
27     count=1K`
28     echo "Eine Datei für Testdaten wurde erstellt."
29 else
30     echo "Es konnte keine Datei erstellt werden."
31     exit 1
32 fi
```

2. Schreiben Sie ein Shell-Skript, das als Kommandozeilenargument einen Dateinamen einliest.

- Die Datei soll das Shell-Skript dahingehend untersuchen, ob es sich um eine Datei, einen Link oder ein Verzeichnis handelt.

- Wenn es sich um eine Datei handelt, soll der Benutzer mit Hilfe von `select` folgende Auswahlmöglichkeiten haben:

- 1) ZIP
- 2) ARJ
- 3) RAR
- 4) GZ
- 5) BZ2
- 6) Alle
- 7) Beenden

- Wählt der Benutzer einen Kompressionsalgorithmus, soll mit diesem die Datei komprimiert werden und der Dateiname entsprechend angepasst werden. Die Dateigröße der originalen und der komprimierten Datei soll das Skript zum Vergleich ausgeben. z.B:

```
Testdatei.txt          <Dateigröße>
Testdatei.txt.rar      <Dateigröße>
```

- Wählt der Benutzer die Auswahlmöglichkeit (Alle), soll das Skript die Datei mit allen Kompressionsalgorithmen komprimieren und die Dateigrößen der originalen und der komprimierten Dateien zum Vergleich ausgeben.

```
Testdatei.txt          <Dateigröße>
Testdatei.txt.zip      <Dateigröße>
Testdatei.txt.arj      <Dateigröße>
Testdatei.txt.rar      <Dateigröße>
Testdatei.txt.gz       <Dateigröße>
Testdatei.txt.bz2      <Dateigröße>
```

```
1 #!/bin/bash
2 #
3 # Skript: archivieren.bat
4 #
5 # Funktion zum komprimieren einer Datei via ZIP
6 zip_packen() {
7     if zip -r $1.zip $1 ; then
8         echo "Die Datei $1 wurde via ZIP komprimiert."
9     else
10        echo "Die Kompression der Datei $1 via ZIP ist
11        fehlgeschlagen."
12    fi
13 }
14 # Funktion zum komprimieren einer Datei via ARJ
15 arj_packen() {
16     if arj a $1.arj $1 ; then
17         echo "Die Datei $1 wurde via ARJ komprimiert."
18     else
19        echo "Die Kompression der Datei $1 via ARJ ist
20        fehlgeschlagen."
```

```
20  fi
21 }
22
23 # Funktion zum komprimieren einer Datei via RAR
24 rar_packen() {
25     if rar a $1.rar $1 ; then
26         echo "Die Datei $1 wurde via RAR komprimiert."
27     else
28         echo "Die Kompression der Datei $1 via RAR ist
29             fehlgeschlagen."
30     fi
31 }
32 # Funktion zum komprimieren einer Datei via GZ
33 gz_packen() {
34     if gzip -c $1 > $1.gz ; then
35         echo "Die Datei $1 wurde via GZ komprimiert."
36     else
37         echo "Die Kompression der Datei $1 via GZ ist
38             fehlgeschlagen."
39     fi
40 }
41 # Funktion zum komprimieren einer Datei via BZ2
42 bz2_packen() {
43     if bzip2 -zk $1 ; then
44         echo "Die Datei $1 wurde via BZ2 komprimiert."
45     else
46         echo "Die Kompression der Datei $1 via BZ2 ist
47             fehlgeschlagen."
48     fi
49 }
50 # Untersuchen ob die als Kommandozeilenargument übergebene
51   Datei existiert
52 if [ ! -e $1 ] ; then
53     # Die Datei existiert nicht.
54     echo "Die Datei $1 existiert nicht."
55     # Das Skript beenden.
56     exit 1
57 fi
58 # Untersuchen ob die Datei ein Verzeichnis ist.
59 if [ -d $1 ] ; then
60     echo "Das Kommandozeilenargument ist ein Verzeichnis."
61     exit
62 elif [ -L $1 ] ; then
63     echo "Das Kommandozeilenargument ist ein symbolischer Link."
64     exit
65 elif [ -f $1 ] ; then
66     echo "Das Kommandozeilenargument ist eine reguläre Datei."
67
68     # Auswahlmöglichkeiten ausgeben.
69     select auswahl in ZIP ARJ RAR GZ BZ2 Alle Beenden
70
71     do
```

```
72  if [ "$auswahl" = "ZIP" ] ; then
73      zip_packen $1
74      ls -lh $1|awk '{print $9,$5}'
75      ls -lh $1.zip|awk '{print $9,$5}' | column -t
76      exit
77  elif [ "$auswahl" = "ARJ" ] ; then
78      arj_packen $1
79      ls -lh $1|awk '{print $9,$5}'
80      ls -lh $1.arj|awk '{print $9,$5}' | column -t
81      exit
82  elif [ "$auswahl" = "RAR" ] ; then
83      rar_packen $1
84      ls -lh $1|awk '{print $9,$5}'
85      ls -lh $1.rar|awk '{print $9,$5}' | column -t
86      exit
87  elif [ "$auswahl" = "GZ" ] ; then
88      gz_packen $1
89      ls -lh $1|awk '{print $9,$5}'
90      ls -lh $1.gz|awk '{print $9,$5}' | column -t
91      exit
92  elif [ "$auswahl" = "BZ2" ] ; then
93      bz2_packen $1
94      ls -lh $1|awk '{print $9,$5}'
95      ls -lh $1.bz2|awk '{print $9,$5}' | column -t
96      exit
97  elif [ "$auswahl" = "Alle" ] ; then
98      zip_packen $1
99      arj_packen $1
100     rar_packen $1
101     gz_packen $1
102     bz2_packen $1
103     ls -lh $1* | awk '{print $9,$5}' | column -t
104     exit
105  else [ "$auswahl" = "Beenden" ]
106     echo "Das Skript wird beendet."
107     exit
108  fi
109 done
110 else
111     exit 1
112 fi
```

3. Testen Sie das Shell-Skript mit der generierten Datei `testdaten.txt`. Was ist das Ergebnis?

Aufgabe 8 (Shell-Skripte, Datei-Browser)

Schreiben Sie ein Shell-Skript, das via `select` einen Datei-Browser realisiert.

- Die Liste der Dateien und Verzeichnisse im aktuellen Verzeichnis soll ausgegeben und die einzelnen Einträge sollen auswählbar sein.

- Wird eine Datei ausgewählt, soll der Dateiname mit Endung, die Anzahl der Zeichen, Wörter und Zeilen sowie eine Information über den Inhalt der Datei ausgegeben werden. z.B:

<Dateiname>.<Dateiendung>

Zeichen: <Anzahl>

Zeilen: <Anzahl>

Wörter: <Anzahl>

Inhalt: <Angabe>

Informationen zur Anzahl der Zeichen, Wörter und Zeilen einer Datei liefert das Kommando wc. Information über den Inhalt einer Datei liefert das Kommando file.

- Wird ein Verzeichnis ausgewählt, soll das Skript in dieses Verzeichnis wechseln und die Dateien und Verzeichnisse im Verzeichnis ausgeben.
- Es soll auch möglich sein, im Verzeichnisbaum nach oben zu gehen (`cd ..`).

```
1 !/bin/bash
2 #
3 # Skript: datei_browser.bat
4 #
5 file=""
6
7 while true
8 do
9     if [ "$file" == ".." ] ; then
10         # In der Verzeichnisstruktur eine Ebene höher gehen
11         cd ..
12     elif [ -d $file ] ; then
13         cd $file          # In ein Verzeichnis wechseln
14     else
15         break
16     fi
17
18     select file in "." * # Dateiauswahlliste ausgeben
19     do
20         break
21     done
22 done
23
24 if [ -f $file ]
25 then
26     echo $file          # Dateinamen mit Endung ausgeben
27     echo "Zeichen: "`wc -m $file | awk '{ print $1 }'`
28     echo "Zeilen:  "`wc -l $file | awk '{ print $1 }'`
29     echo "Wörter:  "`wc -w $file | awk '{ print $1 }'`
30     echo "Inhalt:  "
31     cat $file          # Inhalt der Datei ausgeben
32 fi
```