Lösung von Übungsblatt 9

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. Beschreiben 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. Nennen Sie eine Möglichkeit, um Race Conditions zu vermeiden.

   Durch das Konzept der Semaphore.

Aufgabe 2 (Synchronisation)

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

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

2. Geben Sie an, welche beiden Probleme durch Blockieren entstehen können.

   Verhungern (Starving) und Verklemmung (Deadlock).

3. 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 sicherge-

stellt, dass es keine Überlappung in der Ausführung der kritischen Abschnitte gibt.

4. Kreuzen Sie vier Bedingungen an, die gleichzeitig erfüllt sein müssen, damit ein Deadlock entstehen kann?

☐ Rekursive Funktionsaufrufe	☒ Anforderung weiterer Betriebsmittel
☒ Wechselseitiger Ausschluss	☐ > 128 Prozesse im Zustand `blockiert`
☐ Häufige Funktionsaufrufe	☐ Iterative Programmierung
☐ Geschachtelte `for`-Schleifen	☒ Zyklische Wartebedingung
☒ Ununterbrechbarkeit	☐ Warteschlangen

5. Führen Sie die Deadlock-Erkennung mit Matrizen durch und geben Sie an, ob es zum Deadlock kommt.

$$\text{Ressourcenvektor} = \begin{pmatrix} 8 & 6 & 7 & 5 \end{pmatrix}$$

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

Aus dem Ressourcenvektor und der Belegungsmatrix ergibt sich der Ressourcenrestvektor.

$$\textit{Ressourcenrestvektor} = \begin{pmatrix} 3 & 2 & 6 & 0 \end{pmatrix}$$

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

$$\textit{Ressourcenrestvektor} = \begin{pmatrix} 6 & 3 & 6 & 4 \end{pmatrix}$$

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

$$\textit{Ressourcenrestvektor} = \begin{pmatrix} 6 & 5 & 7 & 5 \end{pmatrix}$$

Nun kann Prozess 1 laufen.

Es kommt nicht zum Deadlock.

Aufgabe 3 (Kommunikation von Prozessen)

1. Geben Sie an, 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 $\Longrightarrow$ Inkonsistenzen.

2. Beschreiben Sie die Aufgabe der 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. Geben Sie an, nach welchem Prinzip Nachrichtenwarteschlangen (Message Queues) arbeiten.
   (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 was passiert, wenn ein Prozess in eine volle Pipe schreiben will.

   Der in die Pipe schreibende Prozess wird blockiert.

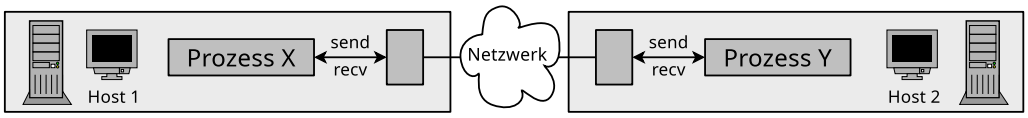
7. Beschreiben Sie was passiert, wenn ein Prozess aus einer leeren Pipe lesen will.

   Der aus der Pipe lesende Prozess wird blockiert.

8. Geben Sie an, welche zwei Arten von Pipes existieren.

   Anonyme Pipes und benannte Pipes.

9. Geben Sie den Namen der im Diagramm gezeigten Form der Interprozesskommunikation an.



   ☐ Gemeinsames Speichersegment
   ☐ Cricket
   ☐ Nachrichtenwarteschlange
   ☐ Anonyme Pipe
   ☒ Sockets
   ☐ Mutex

10. Nennen Sie zwei Vorteile der Form der Interprozesskommunikation in (9).

    Sockets sind eine voll duplexfähige Alternative zu Pipes und gemeinsamen Speichersegmenten. Sockets verwenden Portnummern zur Adressierung und werden vom Betriebssystem verwaltet. Sockets können über das Netzwerk und somit über Betriebssystemgrenzen hinweg verwendet werden.

11. Die Form der Interprozesskommunikation in (9) funktioniert bidirektional.

    ☒ Wahr ☐ Falsch

12. Geben Sie an, welche zwei Arten von Sockets existieren.

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

13. Kommunikation via Pipes funktioniert...
    (Nur eine Antwort ist korrekt!)

    ☐ speicherbasiert ☒ nachrichtenbasiert

14. Kommunikation via Nachrichtenwarteschlangen funktioniert...
    (Nur eine Antwort ist korrekt!)

    ☐ speicherbasiert ☒ nachrichtenbasiert

15. Kommunikation via gemeinsamen Speichersegmenten funktioniert...
    (Nur eine Antwort ist korrekt!)

    ☒ speicherbasiert ☐ nachrichtenbasiert

16. Kommunikation via Sockets funktioniert...
(Nur eine Antwort ist korrekt!)

☐ speicherbasiert ☒ nachrichtenbasiert

17. Geben Sie an, welche drei Formen der Interprozesskommunikation bidirektional funktionieren.

☒ Gemeinsame Speichersegmente ☒ Nachrichtenwarteschlangen
☐ Anonyme Pipes ☐ Benannte Pipes
☒ Sockets

18. Geben Sie an, welche Form der Interprozesskommunikation nur zwischen Prozessen funktioniert die eng verwandt sind.

☐ Gemeinsame Speichersegmente ☐ Nachrichtenwarteschlangen
☒ Anonyme Pipes ☐ Benannte Pipes
☐ Sockets

19. Geben Sie an, welche Form der Interprozesskommunikation über Rechnergrenzen hinweg funktioniert.

☐ Gemeinsame Speichersegmente ☐ Nachrichtenwarteschlangen
☐ Anonyme Pipes ☐ Benannte Pipes
☒ Sockets

20. Geben Sie an, bei welcher Form der Interprozesskommunikation die Daten auch ohne gebundenen Prozess erhalten bleiben.

☒ Gemeinsame Speichersegmente ☒ Nachrichtenwarteschlangen
☐ Anonyme Pipes ☒ Benannte Pipes
☐ Sockets

21. Geben Sie an, bei welcher Form der Interprozesskommunikation das Betriebssystem nicht die Synchronisierung garantiert.

☒ Gemeinsame Speichersegmente ☐ Nachrichtenwarteschlangen
☐ Anonyme Pipes ☐ Benannte Pipes
☐ Sockets

Aufgabe 4 (Kooperation von Prozessen)

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

Ein Semaphor ist eine Zählersperre zur Verwaltung beschränkter Ressourcen und zum Schutz kritischer Abschnitte.

2. Geben Sie die beiden Zugriffsoperationen auf eine Semaphore an.
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 Semaphoren und Blockieren (Sperren und Freigeben).

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

4. Beschreiben Sie was eine binäre Semaphore ist.

   Binäre Semaphore sind Semaphore, die mit dem Wert 1 initialisiert werden und garantieren, dass zwei oder mehr Prozesse nicht gleichzeitig in ihre kritischen Bereiche eintreten können.

5. Beschreiben Sie was ein Mutex ist und seinen Einsatzzweck.

   Wird die Möglichkeit eines Semaphors zu zählen nicht benötigt, kann die vereinfachte Version eines Semaphors, der Mutex, verwendet werden. Mutexe dienen dem Schutz kritischer Abschnitte, auf denen zu jedem Zeitpunkt immer nur ein Prozess zugreifen darf.

6. Geben Sie an, welche Form der Semaphoren die gleiche Funktionalität wie der Mutex.

   Binäre Semaphore.

7. Geben Sie die möglichen Zustände eines Mutex an.

   Die beiden Zustände sind „belegt“ und „nicht belegt“.

8. Geben Sie das Linux/UNIX-Kommando an, das Informationen zu bestehenden gemeinsamen Speichersegmenten, Nachrichtenwarteschlangen und Semaphoren liefert.

   `ipcs`

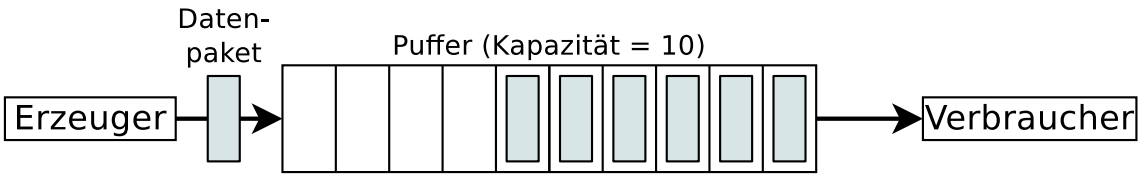
9. Geben Sie das Linux/UNIX-Kommando an, das es ermöglicht, bestehende gemeinsame Speichersegmente, Nachrichtenwarteschlangen und Semaphoren zu löschen.

   `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. Gegenseitiger 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 erzeuger (void) {
  int daten;

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

void verbraucher (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
{
  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);

    }
}
```

```
Auslieferer_X
{
  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);

    }
}
```

```
Auslieferer_Y
{
  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);

    }
}
```

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 (Interprozesskommunikation)

Entwickeln Sie einen Teil eines Echtzeitsystems, das aus vier Prozessen besteht:

1. **Conv**. Dieser Prozess liest Messwerte von A/D-Konvertern (Analog/Digital) ein. Er prüft die Messwerte auf Plausibilität und konvertiert sie gegebenfalls. Wir lassen Conv in Ermangelung eines physischen A/D-Konverters Zufallszahlen erzeugen. Diese müssen in einem bestimmten Bereich liegen, um einen A/D-Konverter zu simulieren.

2. **Log**. Dieser Prozess liest die Messwerte des A/D-Konverters (Conv) aus und schreibt sie in eine lokale Datei.

3. **Stat**. Dieser Prozess liest die Messwerte des A/D-Konverters (Conv) aus und berechnet statistische Daten, unter anderem Mittelwert und Summe.

4. **Report**. Dieser Prozess greift auf die Ergebnisse von Stat zu und gibt die statistischen Daten in der Shell aus.

Bezüglich der Daten in den gemeinsamen Speicherbereichen gelten als Synchronisationsbedingungen:

- **Conv** muss erst Messwerte schreiben, bevor **Log** und **Stat** Messwerte auslesen können.
- **Stat** muss erst Statistikdaten schreiben, bevor **Report** Statistikdaten auslesen kann.

Entwerfen und implementieren Sie das Echtzeitsystem in C mit den entsprechenden Systemaufrufen und realisieren Sie den Datenaustausch zwischen den vier Prozessen einmal mit **Pipes**, **Message Queues** und **Shared Memory mit Semaphore**. Am Ende der praktischen Übung müssen drei Implementierungsvarianten des Programms existieren. Der Quellcode soll durch Kommentare verständlich sein.

Vorgehensweise

Die Prozesse Conv, Log, Stat, und Report sind parallele Endlosprozesse. Schreiben Sie ein Gerüst zum Start der Endlosprozesse mit dem Systemaufruf `fork`. Überwachen Sie mit geeigneten Kommandos wie `top`, `ps` und `pstree` Ihre parallelen Prozesse und stellen Sie die Vater-Kindbeziehungen fest.

Das Programm kann mit der Tastenkombination `Ctrl-C` abgebrochen werden. Dazu müssen Sie einen Signalhandler für das Signal `SIGINT` implementieren. Beachten Sie bitte, dass beim Abbruch des Programms alle von den Prozessen belegten Betriebsmittel (Pipes, Message Queues, gemeinsame Speicherbereiche, Semaphoren) freigegeben werden.

Entwickeln und implementieren Sie die drei Varianten, bei denen der Datenaustausch zwischen den vier Prozessen einmal mit **Pipes**, **Message Queues** und **Shared Memory mit Semaphore** funktioniert.

Überwachen Sie die Message Queues, Shared Memory Bereiche und Semaphoren mit dem Kommando `ipcs`. Mit `ipcrm` können Sie Message Queues, Shared Memory Bereiche und Semaphoren wieder freigeben, wenn Ihr Programm dieses bei einer inkorrekten Beendigung versäumt hat.

Aufgabe 8 (Shell-Skripte, Datenkompression)

1. Schreiben Sie ein Shell-Skript, dass 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.

```
#!/bin/bash
#
# Skript: testdaten_erzeugen.bat
#
# falls Ordner nicht vorhanden, Ordner erzeugen

VERZEICHNIS=/tmp/testdaten
DATEINAME=testdaten.txt

if [ ! -d $VERZEICHNIS ] ; then
  if mkdir $VERZEICHNIS ; then
    echo "Ein Verzeichnis für Testdaten wurde erstellt."
  else
    echo "Es konnte kein Verzeichnis erstellt werden."
  fi
else
  echo "Ein Verzeichnis für Testdaten existiert schon."
  exit 1
fi

if touch `echo "$VERZEICHNIS/$DATEINAME"` ; then
  # Zufallszahl zwischen 128 und 512 erstellen
  ZUFALLSZAHL=`awk -vmin=128 -vmax=512 'BEGIN{srand(); print
   int(min+rand()*(max-min+1))}'`
  # Die Datei mit Nullen füllen
  `dd if=/dev/zero of=$VERZEICHNIS/$DATEINAME bs=$ZUFALLSZAHL
   count=1K`
  echo "Eine Datei für Testdaten wurde erstellt."
else
  echo "Es konnte keine Datei erstellt werden."
  exit 1
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
       fehlgeschlagen."
11   fi
12 }
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
     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
     fehlgeschlagen."
29   fi
30 }
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
     fehlgeschlagen."
38   fi
39 }
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
     fehlgeschlagen."
47   fi
48 }
49
50 # Untersuchen ob die als Kommandozeilenargument übergebene
     Datei existiert
51 if [ ! -e $1 ] ; then
52   # Die Datei existiert nicht.
53   echo "Die Datei $1 existiert nicht."
54   # Das Skript beenden.
55   exit 1
56 fi
57
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 9 (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
```