Lösung von Übungsblatt 9

Aufgabe 1 (Interprozesskommunikation)

1. Was ist ein kritischer Abschnitt?

   Mehrere Prozesse greifen lesend und schreibend auf gemeinsame Daten zu.

2. Was ist eine Race Condition?

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

3. Warum sind Race Conditions schwierig zu lokalisieren und zu beheben?

   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. Wie werden Race Conditions vermieden?

   Durch das Konzept der Semaphore.

Aufgabe 2 (Synchronisation)

1. Welchen Vorteil hat 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. Was ist eine Barriere?

   Eine Barriere synchronisiert die beteiligten Prozesse an einer Stelle.

3. Welche beiden Probleme können durch Blockieren entstehen?

   Verhungern (Starving) und Verklemmung (Deadlock).

4. Was ist der 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. Welche vier Bedingungen müssen gleichzeitig erfüllt sein, 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

6. Kommt es zum Deadlock?
Führen Sie die Deadlock-Erkennung mit Matrizen durch.

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

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

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 (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.

```
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/$DATEINAME"` ; then
22   # Zufallszahl zwischen 128 und 512 erstellen
23   ZUFALLSZAHL=`awk -vmin=128 -vmax=512 'BEGIN{srand(); print
       int(min+rand()*(max-min+1))}'`
24   # Die Datei mit Nullen füllen
25   `dd if=/dev/zero of=$VERZEICHNIS/$DATEINAME bs=$ZUFALLSZAHL
       count=1K`
26   echo "Eine Datei für Testdaten wurde erstellt."
27 else
28   echo "Es konnte keine Datei erstellt werden."
29   exit 1
30 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 4 (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
```