Übungsblatt 6

Aufgabe 1 (Interprozesskommunikation)

1. Beschreiben Sie was ein kritischer Abschnitt ist.
2. Beschreiben Sie was eine Race Condition ist.
3. Erklären Sie war Race Conditions schwierig zu lokalisieren und zu beheben sind.
4. Beschreiben Sie wie Race Conditions vermieden werden.

Aufgabe 2 (Synchronisation)

1. Beschreiben Sie den Vorteil von Signalisieren und Warten gegenüber aktivem Warten (Warteschleife).
2. Beschreiben Sie was eine Barriere ist.
3. Nennen Sie die beiden Probleme, die durch Blockieren entstehen können.
4. Beschreiben Sie den Unterschied zwischen Signalisieren und Blockieren.
5. Markieren Sie die vier Bedingungen, 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

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

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

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

Aufgabe 3 (Kommunikation von Prozessen)

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

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

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.

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

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

8. Nennen Sie die beiden Arten von Pipes.

9. Nennen Sie die beiden Arten von 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!)

    ☐ speicherbasiert ☐ datenstrombasiert
    ☐ objektbasiert ☐ nachrichtenbasiert

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

☐ speicherbasiert ☐ datenstrombasiert
☐ objektbasiert ☐ nachrichtenbasiert

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

☐ speicherbasiert ☐ datenstrombasiert
☐ objektbasiert ☐ nachrichtenbasiert

14. Geben Sie an, welche Form der Interprozesskommunikation bidirektional funktioniert.

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

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

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

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

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

17. Geben Sie an, bei welchen Formen der Interprozesskommunikation die Daten auch ohne gebundenen Prozess erhalten bleiben.

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

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

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

Aufgabe 4 (Kooperation von Prozessen)

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

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

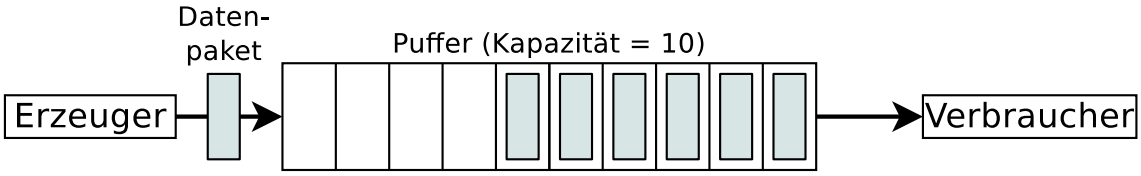
3. Was ist der Unterschied zwischen Semaphoren und Blockieren (Sperren und Freigeben)?

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

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

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;



void erzeuger (void) {
  int daten;
  while (TRUE) {                  // Endlosschleife
    erzeugeDatenpaket(daten);     // erzeuge Datenpaket


    einfuegenDatenpaket(daten);   // Datenpaket in Puffer schreiben


  }
}

void verbraucher (void) {
  int daten;
  while (TRUE) {                  // Endlosschleife


    entferneDatenpaket(daten);    // Datenpaket aus Puffer holen


    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.

```
Lieferant                  Auslieferer_X              Auslieferer_Y
{                          {                          {
  while (TRUE)               while (TRUE)               while (TRUE)
  {                          {                          {

    <Tor durchfahren>;         <Tor durchfahren>;         <Tor durchfahren>;


    <Wareneingang betreten>;   <Warenausgabe betreten>;   <Warenausgabe betreten>;


    <3 Pakete entladen>;       <2 Pakete aufladen>;       <1 Paket aufladen>;


    <Wareneingang verlassen>; <Warenausgabe verlassen>; <Warenausgabe verlassen>;


    <Tor durchfahren>;         <Tor durchfahren>;         <Tor durchfahren>;

    }                          }                          }
}                          }                          }
```

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, 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.
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>
     ```

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 ..`).