Aufgaben zum Thema „Threads und Prozesse“

1. Mit welchem Ziel wurde der Prozeßbegriff für Betriebssysteme eingeführt, warum hat man dieses Ziel angestrebt? Wie lautet eine Begriffsbestimmung für Prozesse in einem möglichst allgemeinen Sinn? Warum und worin unterscheidet sich ein solcher Begriff von dem Begriff, wie er in Betriebssystemen gebraucht wird, und wie lautet eine derartige Begriffsbestimmung? Erläutern Sie diese Begriffsbestimmung! Welche grundlegenden Konsequenzen ergeben sich daraus? Warum unterscheidet man zwischen Single-Thread-Prozessen und Multi-Thread-Prozessen?

2. Erklären Sie Aufgabe und Struktur eines Threadsteuerblocks (TCB)! Welche weiteren Informationen sind für einen Prozeßsteuerblock (PCB) erforderlich?

3. Eine Technik zur Leistungssteigerung eines Rechensystems ist das sog. Auslagern eines Prozesses bzw. Threads („swapping“), d.h. das Speichern der im Hauptspeicher befindlichen prozeßrelevanten Informationen auf einer Platte und das anschließende Freigeben dieses Speicherbereichs.
   a) Welches sind Vor- und Nachteile dieser Technik?
   b) Entwerfen Sie ein Thread-Zustandsmodell (analog zu Folie 5 des Kapitels „Threads“), in dem diese Technik durch einen weiteren Zustand „ausgelagert“ berücksichtigt wird! Welche Zustandsübergänge sind notwendig, welche sind darüber hinaus sinnvoll, welche sollten ausgeschlossen sein? Erläutern Sie dabei auch die verwendeten Zustände und Übergänge!

4. Betrachtet werde ein Mail-Server, der die in einer Liste abgelegte Nachrichten an die jeweiligen Empfänger versenden soll (jede Nachricht habe nur einen einzigen Empfänger). Der Server arbeitet folgendermaßen: er entnimmt der Liste eine Nachricht, extrahiert den Empfänger, ermittelt dessen IP-Adresse, baut die entsprechende Verbindung auf, sendet die Nachricht und schließt die Verbindung.
   a) Beschreiben Sie (in Pseudocode) eine Implementation des Servers durch ein Programm, das als sequentieller Prozeß ausgeführt wird. Dabei werde angenommen, daß stets zu versendende Nachrichten vorliegen.
   b) Warum ist diese Lösung ineffizient? Welche Möglichkeiten gibt es, sie zu verbessern? Hat sie dennoch Vorteile?
   c) Beschreiben Sei eine Implementation, bei der mehrere Threads innerhalb eines Prozesses genutzt werden können! Welche Vor- und Nachteile hat dieses Vorgehen?
   d) Welche Konsequenzen hat es, wenn anstelle der Threads selbst wieder sequentielle Prozesse (mit nur jeweils einem Thread) verwendet werden?

5. In einem System paralleler Prozesse erfolge das Ausdrucken von Dateien folgendermaßen. Ein Prozeß, der eine Datei ausgedruckt haben möchte, schreibt den Dateinamen in ein (genügend großes) Feld; jedes Feldelement enthält also genau einen Dateinamen. Die Einträge erfolgen fortlaufend. Eine Variable *frei* zeigt den nächsten freien Platz an, eine weitere Variable *drucke* die nächste zu druckende Datei. Die Variablen *frei* und *druckliste* können von allen Prozessen genutzt werden. Ein Prozeß *druckprozeß* ermittelt periodisch, ob eine Datei auszudrucken ist, druckt sie gegebenenfalls, streicht sie aus *druckliste* und aktualisiert die Variable *drucke*.

   Erklären Sie an diesem Beispiel die Begriffe Wettlaufsituation – kritischer Abschnitt (kA) – wechselseitiger Ausschluß (wA)!

6. Wie wird wechselseitiger Ausschluß prinzipiell realisiert? Welche allgemeinen Anforderungen muß jede Realisierung erfüllen?

7. Welche Möglichkeiten zur Realisierung des wechselseitigen Ausschlusses gibt es auf Maschinenebene? Welche Vor- und Nachteile haben sie?

8. Erklären Sie, warum die in der Vorlesung angegebene erste Lösung zum wechselseitigen Ausschluß mittels Sperrvariablen (Folie 58) das Problem nicht löst! Worin liegt die prinzipielle Ursache?

9. Im Gegensatz zu Aufg. 8 ist der wechselseitige Ausschluß mehrerer Threads bezüglich eines kritischen Abschnitts mittels einer einfachen Schleife beispielsweise dann korrekt möglich, wenn in der zugrundeliegenden Hardware ein Maschinenbefehl der Form `xchg R adr` bereitgestellt wird, durch den atomar der Inhalt eines Registers `R` und einer Hauptspeicherzelle `adr` miteinander ausgetauscht werden. Diskutieren Sie unter diesem Gesichtspunkt die nachstehende Implementation auf der Basis eines i386-Systems!

```
       .globl  lock, unlock /* Funktionen extern zugaenglich machen */
mutex: .long   0            /* Festlegung einer HS-Zelle mit Namen
                               mutex und Inhalt 0 */

lock:
        movl   $1, %eax
wait:   xchg   %eax, mutex
        cmp    $0, %eax
        jne    wait
        ret

unlock:
        movl   $0, mutex
        ret
```

10. Die für die Lösung des Wettlaufproblems geforderte Bedingung der Lebendigkeit kann folgendermaßen untergliedert werden: Lebendigkeit ist das *Nicht*-Vorliegen von
    - Fernwirkung: Ein Thread außerhalb seines kritischen Abschnitts (und außerhalb der Dienste *entersection*, *leavesection*) behindert den Thread-Ablauf.
    - Ausgrenzung: Ein Thread wird ständig am Eintritt in seinen kritischen Abschnitt gehindert.
    - Verklemmung: Zwei Threads behindern sich gegenseitig am Eintritt in ihren kritischen Abschnitt.

    Diskutieren Sie unter diesem Gesichtspunkt, inwieweit die folgenden fünf Versuche das Wettlaufproblem zwischen zwei Threads T_1, T_2 lösen! (Die Threads sollen die angegebenen Befehlsfolgen jeweils im Zyklus „ewig“ durchlaufen, sofern möglich.)

    Voraussetzung für 1. Versuch: int s = 1;

    1. Versuch:

T_1	T_2
while (s == 2) {}	while (s == 1) {}
/* KA */	/* KA */
s = 2;	s = 1;

    Voraussetzung für 2. - 5. Versuch:

    ```
    #define  false 0
    #define  true 1
    int  s1 = s2 = true;
    ```

    2. Versuch:

T_1	T_2
while (s2 == false) {}	while (s1 == false) {}
s1 = false;	s2 = false;
/* KA */	/* KA */
s1 = true;	s2 = true;

3. Versuch:

T_1

```
s1 = false;
while (s2 == false)
        {}
/* KA */
s1 = true;
```

T_2

```
s2 = false;
while (s1 == false)
        {}
/* KA */
s2 = true;
```

4. Versuch:

T_1

```
s1 = false;
while (s2 == false) {
    s1 = true;
    s1 = false;
}
/* KA */
s1 = true;
```

T_2

```
s2 = false;
while (s1 == false) {
    s2 = true;
    s2 = false;
}
/* KA */
s2 = true;
```

5. Versuch: weitere Voraussetzung: `int next = 1;`

T_1

```
s1 = false;
while (s2 == false) {
    if (next == 2) {
        s1 = true;
        while (next == 2)
            ;
        s1 = false;
    }
}
/* KA */
next = 2;
s1 = true;
```

T_2

```
s2 = false;
while (s1 == false) {
    if (next == 1) {
        s2 = true;
        while (next == 1)
            ;
        s2 = false;
    }
}
/* KA */
next = 1;
s2 = true;
```

11. Untersuchen Sie, ob der folgende Algorithmus die Bedingungen zum Schutz kritischer Abschnitte erfüllt!

```
int  s1 = s2 = next = 1;
```

Thread 1:

```
      s1 = 0;
M1:   if (s2 == 0) {
        if (next == 1) goto M1;
        s1 = 1;
        while (next == 2)
          ;
      }
      /* KA */
      s1 = 1;
      next = 2;
```

Thread 2:

```
      s2 = 0;
M2:   if (s1 == 0) {
        if (next == 2) goto M2;
        s2 = 1;
        while (next == 1)
          ;
      }
      /* KA */
      s2 = 1;
      next = 1;
```

12. Was versteht man unter einem Semaphor? Warum wurde der Begriff eingeführt? Erläutern Sie die in der Vorlesung angegebene Implementation verbal! Welche Vor- und Nachteile hat diese Implementation? Welche Vor- und Nachteile haben Semaphore generell?

13. Die Semaphoroperationen P und V müssen als unteilbare Operationen implementiert werden. Skizzieren Sie einen Ablauf, der andernfalls zu falschen Ergebnissen führt!

14. Geben Sie in C eine Implementation für einen zählenden Semaphor an, dessen Zählvariable `count` auch negative Werte annehmen kann mit folgender Bedeutung:
    - Ist `count` $\geq$ `0`, so gibt `count` die Anzahl der Prozesse an, die noch den kA betreten können.
    - Ist `count` $\leq$ `0`, so gibt `-count` die Anzahl der Prozesse an, die auf das Betreten des kA warten.

    Stellen Sie zunächst einen Programmablaufplan für die Funktionen P und V auf!

15. Beschreiben Sie die Steuerung für ein System, bei dem Fahrzeuge über eine Brücke wollen, für folgende Fälle der maximalen Belastbarkeit der Brücke:
    a) ein Fahrzeug, unabhängig von der Richtung (1 Fahrspur)
    b) ein Fahrzeug je Richtung gleichzeitig (2 Fahrspuren)
    c) drei Fahrzeuge je Richtung gleichzeitig (2 Fahrspuren)
    d) drei Fahrzeuge, unabhängig von der Richtung (1 Fahrspur; die Fahrzeuge aus einer Richtung müssen so lange warten, bis alle Fahrzeuge der Gegenrichtung die Brücke passiert haben)

    unter Verwendung von Semaphoren! Welches Problem tritt in d) auf, wie läßt es sich lösen?

16. a) Erläutern Sie das Erzeuger-Verbraucher-Problem und seine prinzipiellen Lösungsmöglichkeiten!
    b) Der Puffer sei einelementig. Geben Sie eine Lösung dieses speziellen Problems mittels Semaphoren an! Verdeutlichen Sie die Korrektheit Ihrer Lösung anhand eines repräsentativen Ablaufbeispiels!
    c) Inwieweit läßt sich das Problem im Fall b) auch ohne Semaphore (allein auf Maschinenebene) lösen?

17. Ein Ringpuffer werde durch einen Thread A mit Elementen vom Typ `elem_t` gefüllt und durch einen Thread B geleert (gemäß FIFO); der Puffer soll dabei als *n*-elementiges Feld (*n* konstant) implementiert werden.
    a) In welchen Fällen können Konflikte auftreten?
    b) Beschreiben und diskutieren Sie eine Lösung des Problems in C mittels Semaphoren!

18. a) Erläutern Sie das Philosophenproblem! Worin liegt seine Bedeutung?
    b) Betrachtet werde der in der Vorlesung angegebene Lösungsversuch.
    Zeigen Sie, daß
    - weder diese Implementation
    - noch das „Klammern“ der gesamten Anweisungsfolge vom Aufnehmen der ersten bis zum Ablegen der zweiten Gabel
    - noch das einzelne „Klammern“ des Aufnehmens und des Ablegens beider Gabeln

    mit Semaphoren das Problem lösen! Welches ist der entscheidende Gedanke, der zu einer korrekten Lösung führt?

19. Was versteht man unter dem Leser-Schreiber-Problem? Skizzieren Sie mindestens zwei Lösungsansätze! Welches sind die wesentlichen Gedanken der in der Vorlesung angegebenen Lösung?

20. Erklären Sie den Begriff „Prioritätsumkehr“ (priority inversion) in einem System ohne bzw. mit Semaphoren!