

(Prüfungs-)Aufgaben zum Thema Synchronisation

Mit einem **D** gekennzeichnete Aufgaben sind nur für die Prüfung im Diplomstudiengang relevant.

1) Beschreiben Sie die Funktion der P- und V-Operationen für Semaphore.

2) Erklären Sie das Konzept der Mutexe und die P- und V-Operation für Mutexe.

3) Erklären Sie **eine** der in der Vorlesung behandelten Methoden zur Prozeßsynchronisation.

(Wenn Sie mehr als eine beschreiben, wird nur die erste bewertet.)

4)

(2.1) Was versteht man unter einem „kritischen Abschnitt“?

(2.2) Beschreiben Sie die beiden klassischen Operationen auf binären Semaphoren.

(2.3) Wie läßt sich ein kritischer Abschnitt mit Semaphoren sichern?

(2.4) Darf ein Prozeß, der sich im kritischen Abschnitt befindet,

(a) durch Interrupt unterbrochen werden

(b) vom Scheduler verdrängt werden

(c) sich blockieren

(bitte jeweils mit Begründung)?

5)

(1) Zur Lösung des Philosophenproblems werde für jede der Gabeln ein Semaphore namens `gabel[i]` verwendet. Warum ist die folgende Lösung fehlerhaft?

```
#define N 5                                /* Anzahl der Philosophen */

void philosopher (int i)                  /* i: Nummer des Philosophen (0 bis N-1) */
{
    while (TRUE) {
        think();                          /* Der Philosoph denkt */
        P(gabel[i]);                       /* Linke Gabel aufnehmen */
        P(gabel[(i+1) % N]);              /* Rechte Gabel aufnehmen (%=modulo) */
        eat();                             /* Der Philosoph ißt */
        V(gabel[i]);                       /* Linke Gabel zurücklegen */
        V(gabel[(i+1) % N]);              /* Rechte Gabel zurücklegen */
    }
}
```

(2) Modifizieren Sie diese Lösung *entweder* unter Unix (Hinweis: System V IPC) *oder* unter Windows NT, so daß die Synchronisation korrekt wird. (Es kommt nicht auf korrekte Syntax und korrekte Funktionsnamen an.)

6) Semaphore und die dazugehörigen P- und V-Operationen werden verwendet, um den Eintritt in kritische Abschnitte zu synchronisieren. Dabei müssen die P- und V-Operation selbst als kritische Abschnitte implementiert werden, was zunächst wie ein Teufelskreis aussieht.

Erklären Sie, wie P- und V-Operation in einem Betriebssystem implementiert werden können, um den gegenseitigen Ausschluß bei ihrem Aufruf sicherzustellen.

7) Erklären Sie die Begriffe „aktives Warten (busy wait)“ und „passives Warten“. Gehen Sie dabei auch auf die Vor- und Nachteile der beiden Methoden ein.

8) Erklären Sie das Konzept der Semaphore und die P- und V-Operation für Semaphore.

9)

a) Was ist ein „kritischer Abschnitt“?

b) Wie wird der Zugriff auf kritische Abschnitte mit Hilfe von Mutexes synchronisiert? (Erklären Sie dabei auch die Funktionsweise der P- und V-Operation für Mutexe.)

c) Die P- und die V-Operation für Mutexe sind selbst kritische Abschnitte. Wie wird der gegenseitige Ausschluß für die P- und V-Operation üblicherweise sichergestellt?

10) Was versteht man unter aktivem Warten, was unter passivem Warten? Geben Sie jeweils auch die Vor- und Nachteile an.

11) Bearbeiten Sie **entweder** Teil a) **oder** Teil b).

a) Was sind die drei System-V-IPC-Objekte? Beschreiben Sie die Verwendung **eines** dieser Objekte.

b) Was versteht man in Windows NT unter einem Synchronisationsobjekt? Geben Sie drei Beispiele von Synchronisationsobjekten an, von denen eines speziell für die Synchronisation existiert, ein anderes aber auch unabhängig von Synchronisationsaufgaben wichtig ist. Welches Ereignis löst bei Ihren Beispielen das Wecken eines auf das Objekt wartenden Threads aus?

Es folgt die Behandlung von Aufgabenteil ____ :

12) Beschreiben Sie *entweder* für Unix *oder* für Windows NT, wie dort Semaphore implementiert sind (mit den implementierungsspezifischen Details).

Es folgt die Behandlung für _____ .

13) Gegeben sei der folgende (Pseudo-)Code zur Lösung des Erzeuger-Verbraucher-Problems mit Hilfe von Semaphoren:

```
typedef int semaphore;
semaphore mutex = 1; /* Kontrolliert den Zugriff auf den Puffer */
semaphore empty = N; /* Zählt die freien Plätze im Puffer */
semaphore full = 0; /* Zählt die belegten Plätze im Puffer */

void producer (void) {
while (TRUE) { /* Endlosschleife */
produce_item (item); /* Erzeuge etwas für den Puffer */
P (mutex); /* Eintritt in den kritischen Bereich */
P (empty); /* Zahl der leeren Plätze dekrementieren */
enter_item (item); /* In den Puffer einstellen */
V (mutex); /* Kritischen Bereich verlassen */
V (full); /* Zahl der belegten Plätze inkrementieren */
} }

void consumer (void) {
while (TRUE) { /* Endlosschleife */
P (mutex); /* Eintritt in den kritischen Bereich */
P (full); /* Zahl der belegten Plätze dekrementieren */
remove_item (item); /* Aus dem Puffer entnehmen */
V (mutex); /* Kritischen Bereich verlassen */
V (empty); /* Zahl der freien Plätze inkrementieren */
consume_entry (item) /* Verbrauchen */
} }
```

Die Synchronisation enthält einen entscheidenden Fehler. Welches Problem kann bei diesem Code auftreten und wie könnte dieses Problem gelöst werden?

14) Betrachten Sie für das bekannte Philosophenproblem folgende Lösung, bei der pro Gabel ein Mutex `mutex[i]` verwendet wird:

```
#define N 5 /* Anzahl der Philosophen */

void philosopher (int i) /* i: Nummer des Philosophen (0 bis N-1) */
{
while (TRUE) {
think(); /* Der Philosoph denkt */
P (mutex[i]); /* Linke Gabel aufnehmen */
P (mutex[(i + 1) % N]); /* Rechte Gabel aufnehmen */
eat(); /* Der Philosoph ißt */
V (mutex[i]); /* Linke Gabel zurücklegen */
V (mutex[(i + 1) % N]); /* Rechte Gabel zurücklegen */
}
}
```

- i) Erklären Sie, warum diese Lösung falsch ist.
- ii) Manche Betriebssysteme bieten eine Variante der Implementierung von Mutexen und der P-Funktion, mit der obige Lösung sehr leicht zu einer korrekten Lösung gemacht werden kann. Wie muß diese Variante aussehen? (Geben Sie nicht nur ein Stichwort an, sondern auch eine kurze Erklärung.)

15) Geben Sie ein Beispiel für die Nützlichkeit, zwei oder mehr Synchronisationsobjekte atomar anfordern zu können.

16) Beschreiben Sie *entweder* für Unix *oder* für Windows NT *einen* der vom Betriebssystem angebotenen Synchronisationsmechanismen (mit den implementierungsspezifischen Details).

Es folgt die Beschreibung von _____ im Betriebssystem _____ .

18) Eine Applikation, bestehend aus mehreren Threads, muß für einige kritische Abschnitte im Code gegenseitigen Ausschluß sicherstellen. Beschreiben Sie *entweder* für Unix *oder* für Windows NT/2000 einen vom Betriebssystem angebotenen Mechanismus, mit dem dies erreicht werden kann. Beschreiben Sie möglichst alle Details der Funktionalität des von Ihnen gewählten Mechanismus.

Es folgt die Beschreibung von _____ im Betriebssystem _____ .

19)

a) Erklären Sie, wie es zu einer Race Condition kommen kann, wenn ein Thread eine gemeinsam benutzte Variable inkrementiert, und ein anderer Thread diese Variable dekrementiert.

b) Was sind die möglichen Werte dieser Variablen nach Durchführung der beiden Operationen, und welcher Wert ist richtig?

20) (hat nur indirekt was mit Synchronisation zu tun)

a) Beschreiben Sie den Ablauf einer *synchronen* Ein- oder Ausgabe (ohne Berücksichtigung der durch Caching entstehenden Besonderheiten).

b) Beschreiben Sie den Ablauf einer *asynchronen* Ein- oder Ausgabe (ohne Berücksichtigung der durch Caching entstehenden Besonderheiten).

c) Welche Art von Ein-/Ausgabe wird von Read/Write-Befehlen in Hochsprachen und von Bibliotheksroutinen üblicherweise ausgeführt?

d) Wie kann man die „andere“ Art von Ein-/Ausgabe anfordern (ein Stichwort genügt, keine Details)?

21)

Betrachten Sie das in der Vorlesung behandelte Philosophenproblem und die Lösung, bei der jeder Philosoph zunächst seine linke Gabel aufnimmt und anschließend seine rechte Gabel.

a) Warum ist diese Lösung falsch?

b) Wir ändern die obige Lösung jetzt wie folgt ab: *Einer* der Philosophen ist etwas verwirrt und nimmt immer zunächst seine rechte Gabel auf und anschließend seine linke Gabel.

Diskutieren Sie diese Lösung.

(Schreiben Sie keine Romane. Mit der richtigen Idee kann die Antwort sehr kurz ausfallen.)

22)

- a) Erklären Sie was es bedeutet, mehrere Ressourcen atomar anzufordern und zuzuteilen.
- b) Geben Sie je ein Beispiel für eine Situation, in der die Freigabe einer von einem Thread bei seiner Beendigung noch gehaltenen Ressource sinnvoll bzw. unsinnig ist.

23)

Erklären Sie, wie man mit Hilfe von Semaphoren eine Lösung für das Erzeuger-Verbraucher-Problem erstellen kann.

24)

- a) Erklären Sie, wie es zu einer Race Condition kommen kann, wenn zwei Threads beide eine gemeinsam benutzte Variable inkrementieren.
- b) Was sind die möglichen Werte dieser Variablen nach Durchführung der beiden Operationen, und welcher Wert ist richtig?

25)

- a) Erklären Sie, was man unter einer Race Condition versteht.
- b) Geben Sie ein konkretes Beispiel für eine Race Condition (mit Erklärung, wie in Ihrem Beispiel ein „Race“ entstehen kann).

26) Beschreiben Sie, wie in einem Betriebssystem die P- und V-Operationen für Semaphore implementiert sein können (am einfachsten ist wohl eine Art Pseudocode, der auch textuelle Beschreibungen enthält). Berücksichtigen Sie dabei auch, dass die Operationen selbst kritische Abschnitte sind, für die gegenseitiger Ausschluss sichergestellt werden muss.