E 3: Semaphoren und Deadlocks

Lernziele:

- Semaphoren für die Absicherung kritischer Bereiche einsetzen
- Die Beeinflussung von priorisierten Prozessabläufen durch die Absicherung gemeinsamer kritischer Bereiche erkennen
- Reservationssequenzen hinsichtlich ihrer Deadlock-Gefährdung untersuchen.
- Prozessfahrpläne aufstellen und interpretieren.
- Reservationssequenzen abändern, so dass die Deadlock-Gefahr gebannt wird.
- Semaphorfunktionen unter einem verbreiteten Betriebssystem erfolgreich einsetzen

E 3.1: Wirkungsweise der Semaphoroperationen P und V

Vier Prozesse P1..P4 laufen auf einem Mehrprozessorsystem so ab, dass jeder Prozess seinen eigenen Prozessor hat.

a) Ergänzen Sie die Tabelle mit den Inhalten der Semaphor-Warteschlange und dem Wert des Semaphorzählers. Markieren Sie zudem mit Linien die jeweiligen Prozesszustände: *einfache Linie* für "laufend" (running), *Doppellinie* im kritischen Bereich und *gestrichelte Linie* für "blockiert" (waiting). Der Semaphor "krit" sei mit 1 initialisiert.

t

P1	P2	P3	P4	Semaphor-Warteschlange	Zähler
					1 (init)
krit.P					
	krit.P				
krit.V					
		krit.P			
			krit.P		
	krit.V				
		krit.V			
		krit.V*)			
			krit.V		

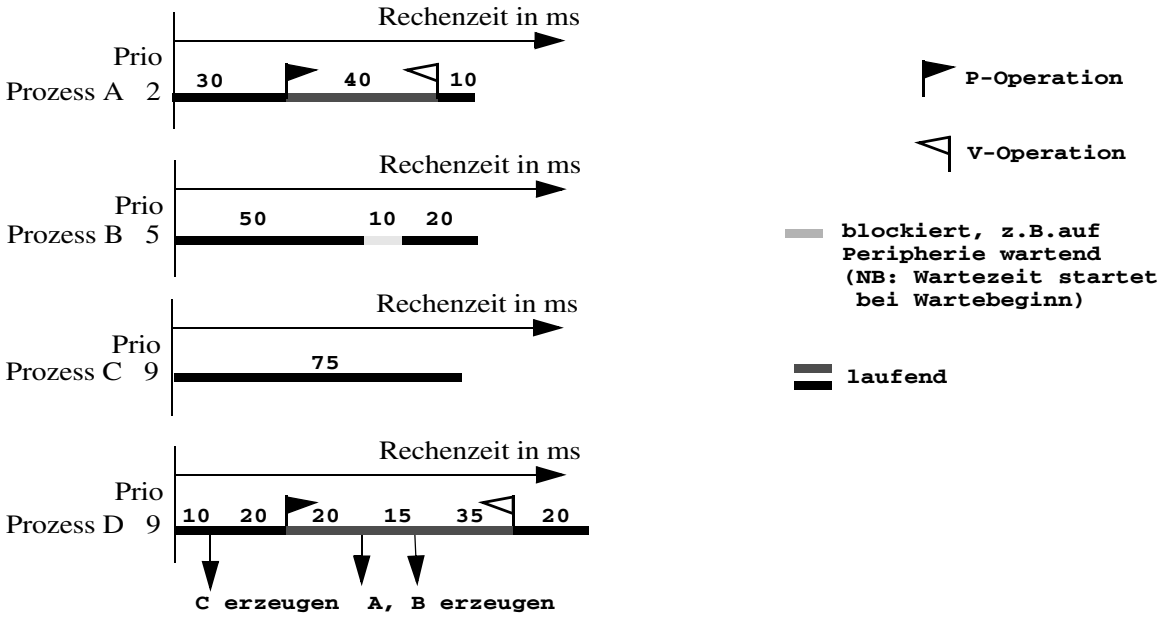
*) Fehler im Programm

Beachten Sie, dass in dieser Tabelle die Zeitachse von oben nach unten abläuft.

b) Überlegen Sie sich, was der mit *) bezeichnete Programmierfehler für Folgen haben könnte.

E 3.2 Parallele Prozesse und Semaphore

In einem Einprozessorsystem werden vier Prozesse A, B, C, D quasiparallel ausgeführt, die für sich alleine wie folgt ablaufen würden:



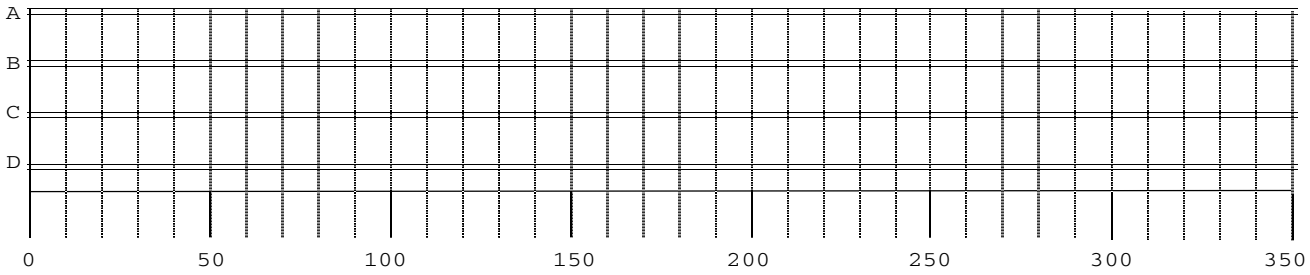
Die Prozesse A und D verwenden einen gemeinsamen Semaphor **`krit`**, der mit 1 initialisiert sei.
Die Prozesse weisen verschiedene Scheduling-Prioritäten auf (2 hohe Prio, 9 tiefe Prio). Es komme für die Prozessorzuteilung eine ML-Strategie zur Anwendung. Innerhalb der gleichen Priorität wird zusätzlich die RR-Strategie angewendet. Die Zeitscheibe weise eine Dauer von 20 ms auf
(1 Zeitquantum = 1 Zeitscheibe).

Beachten Sie folgende Ursachen für ein Rescheduling (*priority based, preemptive scheduling*):

- P-, V-Operation (falls zutreffend)
- Allgemein: Sobald ein Prozess blockiert wird (oder terminiert).
- Wenn ein Prozess gestartet wird.
- Wenn ein wartender Prozess geweckt wird (siehe Prozess B).
- Wenn eine Zeitscheibe beendet wird. (Regel für Round-Robin-Umschaltung: laufender Prozess muss Prozessor an ablaufbereiten Prozess gleicher Priorität abgeben, wenn sein Zeitquantum abgelaufen ist. Dies ist hier am Ende jeder Zeitscheibe der Fall. Jeder ablaufbereite Prozess besitzt ein Zeitquantum von 1.)

Zeichnen Sie den zeitlichen Ablauf in untenstehendes Schema ein und diskutieren Sie die Lösung.

Achtung: Am Anfang werde nur der Prozess D gestartet.



E 3.3: Prozessfahrplan.

Untersuchen Sie folgende Reservationssequenzen hinsichtlich möglicher Deadlocks. Zeichnen Sie dazu je einen Prozess-Fahrplan für die beiden folgenden Fälle (Semaphoren für die Absicherung initialisiert, Wert = ?):

a)

Prozess A	Prozess B
`P(drucker)`	`P(drucker)`
`P(file1)`	`P(diskette)`
`P(diskette)`	`P(file2)`
`V(diskette)`	`V(drucker)`
`P(file2)`	`P(file1)`
`V(drucker)`	`V(file2)`
`V(file2)`	`V(file1)`
`V(file1)`	`V(diskette)`

b)

Prozess X	Prozess Y
`P(a);`	`P(b);`
`P(c);`	`P(a);`
`V(a);`	`P(c);`
`P(d);`	`V(c);`
`P(b);`	`V(a);`
`V(b);`	`P(d);`
`V(c);`	`V(b);`
`V(d);`	`V(d);`

E 3.4: Prozesssynchronisation

Die drei abgebildeten Prozesse 1 bis 3 benutzen gemeinsame, mit 1 initialisierte Semaphoren.

```
Prozess1                Prozess2                Prozess3

loop                    loop                    loop
    P(region)               P(krit)                 P(leit)
    P(disk)                 P(region)               P(disk)
    V(disk)                 V(krit)                 V(leit)
    V(region)               P(leit)                 V(disk)
endloop;                    V(leit)                 endloop;
                            V(region)
                        endloop;
```

a) Funktioniert das Zusammenwirken der 3 Prozesse problemlos? (Allfällige Problemsituation aufzeigen)

b) Was müsste geändert werden?

E 3.5: Begriffe.

a) Was sind die Unterschiede zwischen *Blockieren*, *Verklemmen* und *Verhungern*?

b) Worin unterscheiden sich aktives Warten (*busy wait*) und passives Warten?

E 3.6: Deadlock-sichere Systeme.

Ein Computersystem besitze sechs Ein-/Ausgabekanäle. Es werden *n* Prozesse ausgeführt, wobei ein Prozess jeweils zwei Kanäle benötigt, die er bei Bedarf reserviert (auch überlappend). Wir gehen davon aus, dass alle sechs Kanäle gleichwertig sind, d.h. es keine Rolle spielt welche zwei Kanäle ein Prozess belegt. Konnte ein Prozess zwei Kanäle reservieren, so gibt er diese nach einer gewissen Zeit (=Gebrauchszeit) wieder frei.
Wie gross darf *n* maximal sein für ein sicheres System?

E 3.7: Deadlock Vermeidung.

Ein Transaktionssystem einer Bank benutzt für jede der vielen parallel möglichen Überweisungen von einem Konto A nach einem Konto B einen eigenen Prozess. Es existieren eine Vielzahl von Konten, zwischen denen solche A nach B Überweisungen möglich sind. Lese-/Schreibzugriffe auf ein Konto müssen daher vor gleichzeitigem Zugriff durch mehr als einen Prozess geschützt werden.

a) Wird pro Konto je ein Semaphor zur Absicherung verwendet, so können Deadlocks entstehen. Zeigen Sie eine mögliche Deadlock-Situation auf.

b) Nach welchem Schema müssten die Zugriffe ausgeführt werden, damit keine Deadlocks auftreten können?
Hinweis: Lösungen, die zuerst ein Konto reservieren, modifizieren, freigeben und dann das andere Konto bearbeiten, sind gefährlich, da bei einer Systemstörung Beträge zwischen den Konten verloren gehen können. Suchen Sie daher nach alternativen Lösungen.

E 3.8 Semaphore unter Windows im Einprozessorsystem (optional, bei Interesse und Zeit).

Als Vorlagedatei für diese Übung dient **`semaphore.c`**. Dieses Programm erzeugt zwei Worker Threads. Der eine Thread inkrementiert beide globalen Variablen "CntA" und "CntB". Der andere Thread inkrementiert "CntA" und dekrementiert "CntB". Beide Threads enden nach je 10.000.000 Zyklen, so dass "CntA" den Wert 20.000.000 und "CntB" den Wert 0 aufweisen sollte.

Der Primary Thread (verkörpert durch die **`main()`**-Funktion) zeigt fortlaufend den aktuellen Stand von "CntA" und "CntB" an.

a) Unverändertes Beispielsprogramm:
- Programm **`semaphore.c`** auf der Kommandozeile übersetzen:
  `gcc semaphore.c -o semaphore`
- Code von **`semaphore.c`** studieren
- Mehrmals Programm starten und Endwerte notieren
- Was ist der Grund für die unerwarteten Werte? Welches bzw. welche Probleme liegen vor?

.....

.....

.....

.....

.....

b) Lösung mittels unterschiedlicher Thread-Prioritäten.
Die Thread-Priorität entscheidet, welcher Thread die CPU zugeteilt erhält, wenn mehrere Threads ablaufbereit sind.

- Ändern Sie das Programm ab, so dass Thread 1 eine höhere Thread-Priorität als Thread 2 hat.
  Tipp: Funktion **`SetThreadPriority()`** verwenden (Hilfeinfo via MSDN Library).
- Abgeändertes Programm übersetzen
- Beobachten Sie den Programmablauf
- Ist das Problem nun gelöst? Wenn nein, warum nicht?

.....

.....

.....

.....

c) Lösung mit Semaphoren

- Ändern Sie das Programm ab, so dass die kritischen Bereiche mit Semaphoren abgesichert sind. Reduzieren Sie die Anzahl Durchläufe in Thread 1 und Thread 2 von "1e7" auf "1e6" (1.000.000).
  Tipp: Für jede gemeinsame Ressource einen eigenen Semaphor benutzen.
- Beobachten Sie den Programmablauf
- Entsprechen die Resultate den Erwartungen; wenn nein: haben Sie die Semaphoren korrekt eingesetzt?
- Wieso wurde oben empfohlen, für jede gemeinsame Ressource einen eigenen Semaphor zu benutzen? Oder umgekehrt gefragt, was wäre der Nachteil einer gemeinsamen Absicherung?

.....

.....

.....

.....

d) Semaphorvarianten unter Windows (*freiwillige Aufgabe; bei Interesse*)
Welche Variationen des Semaphors existieren unter der Win32-API und was sind die Unterschiede?

.....

.....

.....

.....

.....

Musterlösung

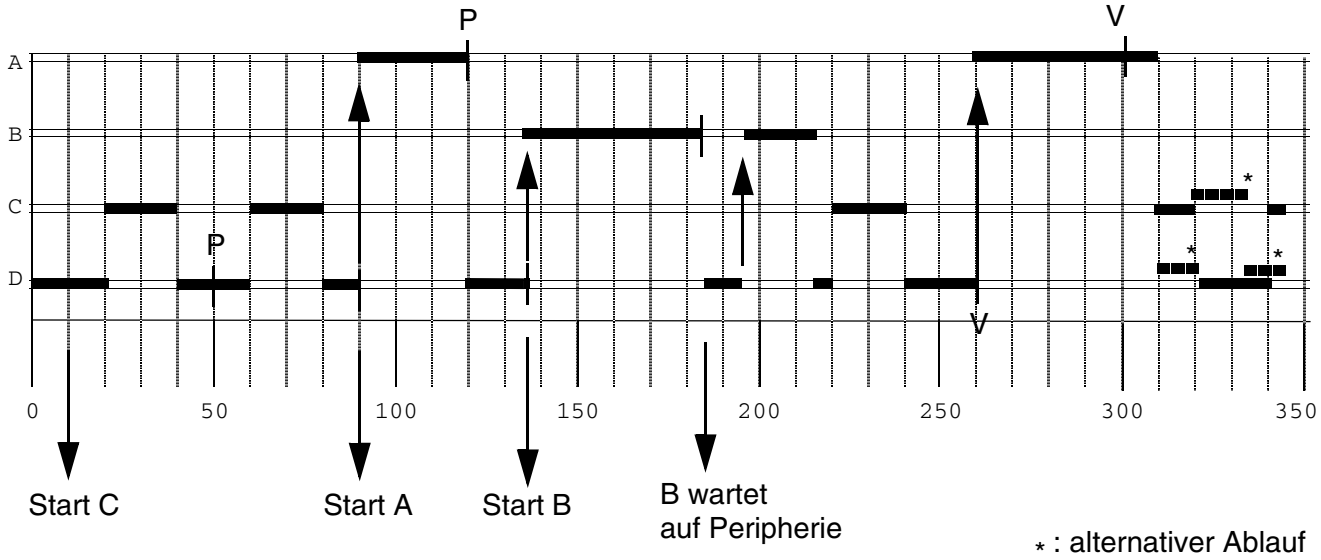
E 3.1: Wirkungsweise der Semaphoroperationen P und V.

a)

P1	P2	P3	P4	Semaphor-WarteschlangeS	Zähler
krit.P					1(init)
					0
	krit.P			P2	0
krit.V					0
		krit.P		P3	0
			krit.P	P3, P4	0
	krit.V			P4	0
		krit.V			0
		krit.V*)			1
					1
			krit.V		(2)

b) Der kritische Bereich ist nicht mehr geschützt! Sowohl P4, als auch der Semaphor sind im Besitz einer Marke, wodurch ein zweiter Prozess den kritischen Bereich ungehindert betreten kann.

E 3.2 Parallele Prozesse und Semaphore



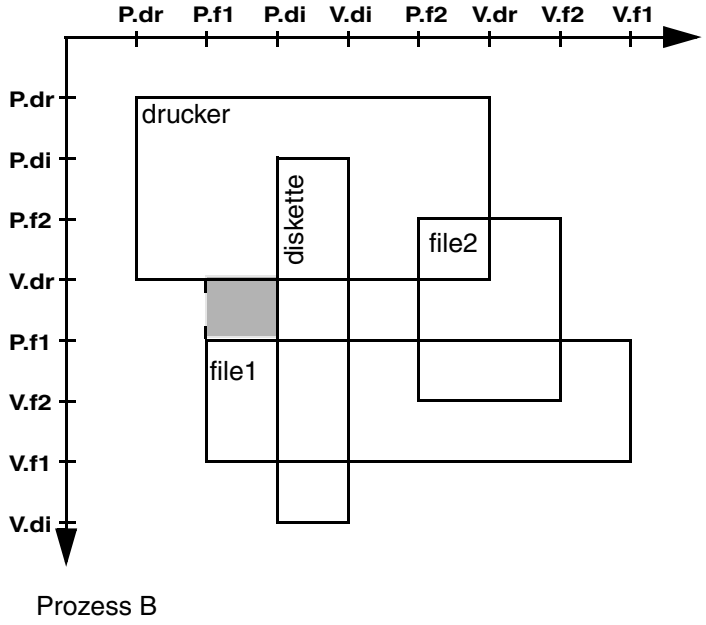
NB: Verdrängte Prozesse, deren Zeitquantum noch nicht aufgebraucht ist, werden vorne und nicht hinten in der Bereitliste eingeordnet.

Diskussion der Lösung:

- Prozess A wird ungebührlich lange blockiert, da Prozess D den kritischen Bereich lange belegt. Zusätzlich stehlen die Prozesse C und B Prozessorzeit (-> Problem der Prioritätsumkehrung bzw. *priority inversion*).
- Verbesserungsmöglichkeiten für Ablauf des Prozesses A:
  Verwendung eines Semaphors mit Prioritätsvererbung (*priority inheritance*); falls kein derartiger Semaphortyp verfügbar: vor Aufruf von P in D die Priorität (9) von D auf die Priorität (2) von A anheben. Nach dem Aufruf von V die Priorität wieder auf den ursprünglichen Wert (9) absenken (*Priority-Ceiling*-Methode).
  Ist auch diese Möglichkeit mangels eines entsprechenden Systemaufrufs ("Priorität setzen") nicht verfügbar, so kann Prozess D den Scheduler mit *lock()/unlock()* für die Dauer des kritischen Bereiches blockieren .

E 3.3:

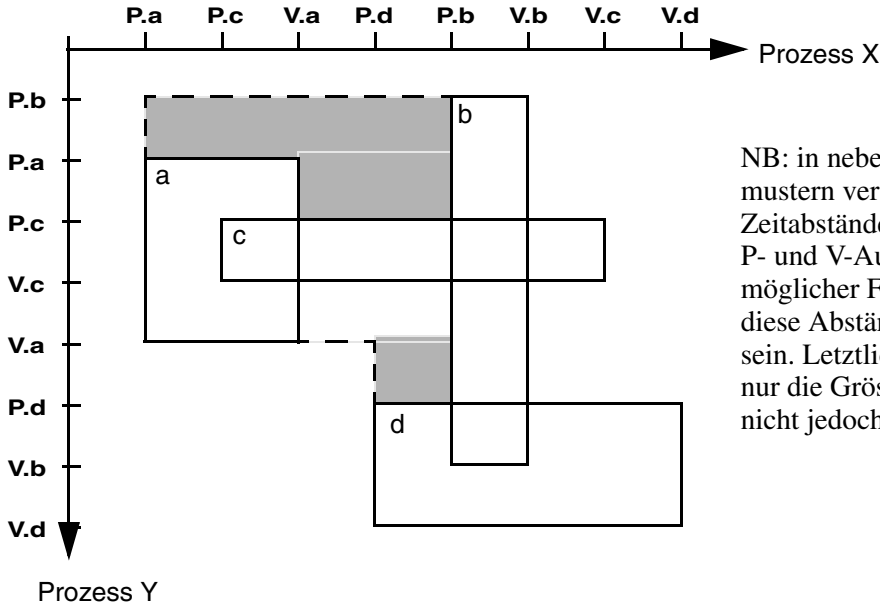
a)



Jeder Semaphor ist mit einer Marke zu initialisieren (Zähler = 1).

Diskussion der Resultate:
Wird die gestrichelte Linie überschritten, so kommt es zu einem Deadlock. Als Folge des erstellten Prozessfahrplanes wäre in einem weiteren Schritt die Reservationsreihenfolge zu modifizieren.

b)



NB: in nebenstehenden Lösungsmustern verstreichen einheitliche Zeitabstände zwischen den einzelnen P- und V-Aufrufen. Dies ist nur ein möglicher Fall. In der Praxis werden diese Abstände sehr unterschiedlich sein. Letztlich beeinflusst dies aber nur die Grösse der Gefahrenflächen, nicht jedoch ihr Auftreten.

E 3.4: a) Verschiedene Möglichkeiten zur Überprüfung:

- nach Deadlock-Abläufen suchen -> Nachteil: ev. findet man nicht alle Abläufe
- dreidimensionaler Prozessfahrplan (ergibt drei sich nicht schneidende Quadrate im Raum)
- sich gedanklich überlegen, ob zyklische Reservationen vorliegen
- eine Ressourcennumerierung suchen, bei welcher alle drei Prozesse die Ressourcen nur nach aufsteigenden Nummern reservieren. Falls eine auffindbar: es herrscht keine Deadlock-Gefahr. Achtung: *Umkehrschluss gilt nicht* (siehe Beispiele unten): Falls eine Nummerierung zu Nichteinhalten der Reihenfolge führt, kann *nicht* geschlossen werden, dass ein Deadlock möglich ist.

z.B. Reihenfolge 1:1: region, 2: disk, 3: leit

Prozess 1	Prozess 2	Prozess 3	
1	1	3	
2	3	2	Schluss:
			Deadlock-Gefahr ; dieser Schluss ist *nicht* zulässig!

z.B. Reihenfolge 2: 1: region, 2: leit, 3: disk

Prozess 1	Prozess 2	Prozess 3	
1	1	2	
3	2	3	Schluss:
			keine Deadlock-Gefahr; dieser Schluss ist zulässig!

NB: Nur wenn alle möglichen Nummerierungsreihenfolgen nicht die Regel der aufsteigenden Nummerierung erfüllen, dann herrscht Deadlockgefahr.

b) nichts, da keine Deadlockgefahr herrscht.

E 3.5: Begriffe.

a) *Blockierung*: Eine Blockierung liegt z.B. dann vor, wenn ein Prozess ein Betriebsmittel reserviert und dann "vergisst" wieder freizugeben (fehlende V()-Operation auf schützendem Semaphor). Alle anderen Prozesse, die das Betriebsmittel nutzen wollen, werden durch ihn blockiert. Diese Blockie-

rung ist dauerhaft und unterscheidet sich damit von den Blockierzuständen, die Prozesse vorübergehend einnehmen, wenn sie auf etwas warten müssen, das nach einer gewissen Zeit eintrifft.
Verklemmung (Deadlock): Zwei oder mehr Prozesse haben je ein Betriebsmittel reserviert und wollen je ein weiteres Betriebsmittel belegen, das bereits von einem anderen Prozess reserviert ist. Da keiner der Prozesse sein bereits reserviertes Betriebsmittel vorzeitig freigeben will, warten alle Prozesse ewig.
Verhungern: Ein ablaufbereiter Prozess erhält nie den Prozessor zugeteilt, da stets andere ablaufbereite Prozesse mit höherer Priorität ihm vorgezogen werden.

b) Beim *aktiven Warten* wird andauernd Rechenzeit verbraucht, um zu prüfen, ob die Wartebedingung erfüllt ist. Beim *passiven Warten* wird der Prozess vom Betriebssystem solange schlafen gelegt, bis das Warteereignis eintrifft.

E 3.6: Deadlock-sichere Systeme.

Das System ist sicher mit maximal 5 Prozessen, da dann immer mindestens 1 Prozess weiterfahren kann und die von ihm belegten Betriebsmittel wieder freigibt.

E 3.7: Deadlock Vermeidung

a) Beispiel für 2 Konten X und Y mit denen Transaktionen ausgeführt werden. Reservationsreihenfolge sei stets das Lastkonto zuerst und dann das Gutschriftskonto nachfolgend.

Transaktion 1	Transaktion 3
P(X)	P(Z)
P(Z)	P(X)
V(X)	V(Z)
V(Z)	V(X)

b) Jedes Konto bekommt eine Nummer. Für eine Überweisung wird immer zuerst das Konto mit der niedrigeren Nummer reserviert und erst anschliessend das mit der höheren Nummer. Damit wird die Deadlock-Bedingung des zyklischen Wartens (*circular wait*) durchbrochen.

E 3.8 Semaphore unter Windows

a) Die Ausführung der Threads kann an beliebigen Programmstellen abwechseln. Dadurch ist es möglich, dass CntA und CntB nur teilweise aktualisiert sind, wenn eine Thread-Umschaltung stattfindet (Problem der verlorenen Aktualisierung, *lost update problem*). Damit sind CntA und CntB unter Umständen nicht mehr konsistent bzw. die auf ihnen ausgeführten Operationen nicht unteilbar.

b) Der Thread mit höherer Priorität unterbricht den Thread mit tieferer Priorität immer noch an beliebiger Stelle. Daher gibt es immer noch Operationen auf CntA und CntB, die nicht unteilbar sind.

c) Bitte Lösungsmuster beachten: **`loesungen\semaphore.c`**

d) *Critical Section Object*: Binärer Semaphor, sehr schnell, nur zwischen Threads innerhalb des gleichen Prozesses anwendbar

Mutex Object: Binärer Semaphor, auch prozessübergreifend verwendbar

Semaphore Object: Zählsemaphor, prozessübergreifend verwendbar

Event Object: Zählsemaphor, neben P-,V-Operationen auch weitere Teilfunktionen verfügbar (Anwendung: Signalisierung von Ein-/Ausgabeereignissen von Treiber an Applikation