Technische Universität Braunschweig

Institut für Betriebssysteme und Rechnerverbund

Betriebssysteme – Übung

6T: Synchronisation

Signe Rüsch, Wintersemester 2018

Übersicht

- 6.1 Kritischer Abschnitt
- 6.2 Synchronisationsalgorithmen
- 6.3 Synchronisation mit Mutex und Semaphoren
- 6.4 Anwendungsfall: Erzeuger-Verbraucher-Problem

Übersicht

a) Kritischer Abschnitt

a) Was versteht man unter einem **kritischen Abschnitt**?

a) Kritischer Abschnitt

Kritischer Abschnitt:

- Gegeben seien n Prozesse $P_0, P_1, \ldots, P_{n-1}$ in einem System
- Jeder dieser Prozesse verfügt über eine Anweisungsfolge, die einen **kritischen Abschnitt** darstellt
- Innerhalb des kritischen Abschnitts verändert ein Prozess Variablen und/oder Dateien, die auch von **anderen** Prozessen genutzt und verändert werden
- Nur **ein** Prozess soll jeweils in der Lage sein, den kritischen Abschnitt zu betreten

a) Kritischer Abschnitt

Kritischer Abschnitt:

- Gegeben seien n Prozesse $P_0, P_1, \ldots, P_{n-1}$ in einem System
- Jeder dieser Prozesse verfügt über eine Anweisungsfolge, die einen **kritischen Abschnitt** darstellt
- Innerhalb des kritischen Abschnitts verändert ein Prozess Variablen und/oder Dateien, die auch von **anderen** Prozessen genutzt und verändert werden
- Nur **ein** Prozess soll jeweils in der Lage sein, den kritischen Abschnitt zu betreten

→ Eine Lösung für das Problem des kritischen Abschnitts ist ein Protokoll, welches genau die zuletzt beschriebene Eigenschaft erzwingt

b) Benötigte Eigenschaften

b) Welche **Eigenschaften** sollten sinnvolle Lösungen zur Behandlung eines kritischen Abschnitts haben?

b) Benötigte Eigenschaften

Eine Lösung für das Problem des kritischen Abschnitts muss folgende Anforderungen erfüllen:

b) Benötigte Eigenschaften

Eine Lösung für das Problem des kritischen Abschnitts muss folgende Anforderungen erfüllen:

- Gegenseitiger Ausschluss
  - Wenn Prozess P_i sich im kritischen Abschnitt befindet, darf kein anderer Prozess seinen kritischen Abschnitt betreten

b) Benötigte Eigenschaften

Eine Lösung für das Problem des kritischen Abschnitts muss folgende Anforderungen erfüllen:

- Gegenseitiger Ausschluss
  - Wenn Prozess P_i sich im kritischen Abschnitt befindet, darf kein anderer Prozess seinen kritischen Abschnitt betreten
- Fortschritt
  - Kein Prozess, der außerhalb seines kritischen Abschnitts läuft, darf andere Prozesse blockieren

b) Benötigte Eigenschaften

Eine Lösung für das Problem des kritischen Abschnitts muss folgende Anforderungen erfüllen:

- Gegenseitiger Ausschluss
  - Wenn Prozess P_i sich im kritischen Abschnitt befindet, darf kein anderer Prozess seinen kritischen Abschnitt betreten
- Fortschritt
  - Kein Prozess, der außerhalb seines kritischen Abschnitts läuft, darf andere Prozesse blockieren
- Beschränkte Wartezeit
  - Kein Prozess sollte ewig darauf warten müssen, in seinen kritischen Abschnitt einzutreten

c) Semaphor vs. Mutex

c) Was ist der Unterschied zwischen **Semaphor** und **Mutex**?

c) Semaphor vs. Mutex

Semaphor

- Datenstruktur/Variable bestehend aus…
  - Positiver Ganzzahl
  - `Increment`-Operation: `up(<varname>)` oder `post(<varname>)`
  - `Decrement`-Operation: `down(<varname>)` oder `wait(<varname>)`
  - Prozess wird geblockt, wenn er versucht, eine `Decrement`-Operation auf einem Semaphor mit einem Wert von `0` anzuwenden
- "Wert prüfen", "Wert ändern" und "geblockt werden" sind **atomare** Aktionen
  - Prozesse dürfen bei diesen Operationen nicht unterbrochen werden
  - ➔ Damit wird sichergestellt, dass zwischen Prüfen und blockiert werden keine Änderung der Ganzzahl stattfinden kann

c) Semaphor vs. Mutex

Mutexe (Mutual Exclusion)

- Im Prinzip wie ein Semaphor, der nur die Werte `1` und `0` annehmen kann
- Zu jedem Zeitpunkt kann nur **ein** Prozess in den kritischen Abschnitt eintreten
- **Locking**-Mechanismus vs. **Signaling**-Mechanismus von Semaphoren

d) Synchronisation ohne aktives Warten

d) Welche Möglichkeiten gibt es für die Synchronisation **ohne** aktives Warten? Welche Realisierungen gibt es?

d) Synchronisation ohne aktives Warten

d) Welche Möglichkeiten gibt es für die Synchronisation **ohne** aktives Warten? Welche Realisierungen gibt es?

- **Falsche** Antwort:
  - Semaphor
  - Mutex
- Erklärung:
  - Semaphor und Mutex sind von der Implementierung abstrahierte Konstrukte, die das Betriebssystem zur Verfügung stellt

  ➔ Sie können auch durch aktives Warten realisiert sein!

d) Synchronisation ohne aktives Warten:

Interrupt-Sperre

- Blockade des Interrupt-Mechanismus in der CPU
- Beispiel (Intel x86):
  - `CLI (CLear Interrupts)` zum **Sperren** von Interrupts
  - `STI (SeT Interrupts)` zum **Freigeben** von Interrupts

Hardware-gestützte atomare Operationen

- Nicht unterbrechbare (atomare) Vergleich- und Schreiboperation
- Hilfreich zur Implementierung von Schlossalgorithmen
- Beispiel (Motorola 64K): `TAS (Test And Set)`
- Beispiel (Intel x86): `XCHG (eXCHanGe)`

e) User Mode

e) Können die Verfahren aus d) in jedem Fall problemlos durch einen **Benutzerprozess** (`User-Mode`) ausgeführt werden?

e) User Mode

Interrupt-Sperre

- Dürfen **nicht** durch Benutzer ausgeführt werden, weil sonst die Gefahr besteht, dass die Sperre nicht mehr aufgehoben wird
  - Unterbrechung nicht mehr möglich
  - “Denial-of-Service” → Blockade des gesamten Systems

→ Privilegierte Operation (nur vom Betriebssystem ausführbar)

e) User Mode

Interrupt-Sperre

- Dürfen **nicht** durch Benutzer ausgeführt werden, weil sonst die Gefahr besteht, dass die Sperre nicht mehr aufgehoben wird
  - Unterbrechung nicht mehr möglich
  - “Denial-of-Service” → Blockade des gesamten Systems

→ Privilegierte Operation (nur vom Betriebssystem ausführbar)

Hardware-gestützte atomare Operationen

- Zwischen zwei atomaren Operation kann der Benutzerprozess unterbrochen werden
  - Sie haben dadurch keine Auswirkungen auf den Rest des Systems

→ Können durch Benutzerprozesse ausgeführt werden

Übersicht

Synchronisationsalgorithmen

Gegeben sei ein Algorithmus für gegenseitigen Ausschluss…

- Zwei Prozesse (P_i und P_j)
- Eine gemeinsame Ressource
- Primitive Datentypen werden atomar geschrieben

→ Wenn beide Prozesse auf die gleiche Variable schreibend zugreifen, so setzt sich der Schreibvorgang eines Prozesses durch!

Synchronisationsalgorithmen

Vor dem Start werden folgende Datenstrukturen initialisiert:

```
//zeigt an, ob Prozesse bereit sind fuer Eintritt in krit. Bereich
int marker[2] = {0, 0};

//zeigt an, wer als naechstes den krit. Bereich betreten darf
int action = -1;
```

Synchronisationsalgorithmen

Algorithmus aus der Sicht von P_i

- wobei `i=0` und `j=1`:

```
int marker[2] = {0, 0};
int action = -1;
```

```
1  while (1) {
2    marker[i] = 1;
3    action = j;
4    while (marker[j] && (action == j)) {
5                        ;
6    }
7    /* critical section */
8    marker[i] = 0;
9    /* remainder section */
10 }
```

Synchronisationsalgorithmen

Algorithmus aus der Sicht von P_i

- wobei `i=0` und `j=1`:

```
int marker[2] = {0, 0};
int action = -1;
```

```
1  while (1) {
2    marker[i] = 1;          // i setzt sich selbst auf bereit
3    action = j;
4    while (marker[j] && (action == j)) {
5                            ;
6    }
7    /* critical section */
8    marker[i] = 0;
9    /* remainder section */
10 }
```

Synchronisationsalgorithmen

Algorithmus aus der Sicht von P_i

- wobei `i=0` und `j=1`:

```
int marker[2] = {0, 0};
int action = -1;
```

```
1  while (1) {
2    marker[i] = 1;          // i setzt sich selbst auf bereit
3    action = j;             // i gibt j den Vorzug
4    while (marker[j] && (action == j)) {
5                            ;
6    }
7    /* critical section */
8    marker[i] = 0;
9    /* remainder section */
10 }
```

Synchronisationsalgorithmen

Algorithmus aus der Sicht von P_i

- wobei `i=0` und `j=1`:

```
int marker[2] = {0, 0};
int action = -1;
```

```
1  while (1) {
2    marker[i] = 1;          // i setzt sich selbst auf bereit
3    action = j;             // i gibt j den Vorzug
4    while (marker[j] && (action == j)){//Solange j bereit & im Vorzug
5                            ;   // ...wartet i aktiv, bis j fertig ist
6    }
7    /* critical section */
8    marker[i] = 0;
9    /* remainder section */
10 }
```

Synchronisationsalgorithmen

Algorithmus aus der Sicht von P_i

- wobei `i=0` und `j=1`:

```
int marker[2] = {0, 0};
int action = -1;
```

```
1  while (1) {
2    marker[i] = 1;          // i setzt sich selbst auf bereit
3    action = j;             // i gibt j den Vorzug
4    while (marker[j] && (action == j)){//Solange j bereit & im Vorzug
5                            ;   // ...wartet i aktiv, bis j fertig ist
6    }
7    /* critical section */   // krit. Bereich betreten
8    marker[i] = 0;
9    /* remainder section */
10 }
```

Synchronisationsalgorithmen

Algorithmus aus der Sicht von P_i

- wobei `i=0` und `j=1`:

```
int marker[2] = {0, 0};
int action = -1;
```

```
1  while (1) {
2    marker[i] = 1;          // i setzt sich selbst auf bereit
3    action = j;             // i gibt j den Vorzug
4    while (marker[j] && (action == j)){//Solange j bereit & im Vorzug
5                            ;   // ...wartet i aktiv, bis j fertig ist
6    }
7    /* critical section */   // krit. Bereich betreten
8    marker[i] = 0;           // krit. Bereich verlassen
9    /* remainder section */
10 }
```

a) Synchronisationsalgorithmen

a) Handelt es sich bei dem Algorithmus um eine valide Lösung zur Koordination eines kritischen Abschnitts zwischen zwei Prozessen?

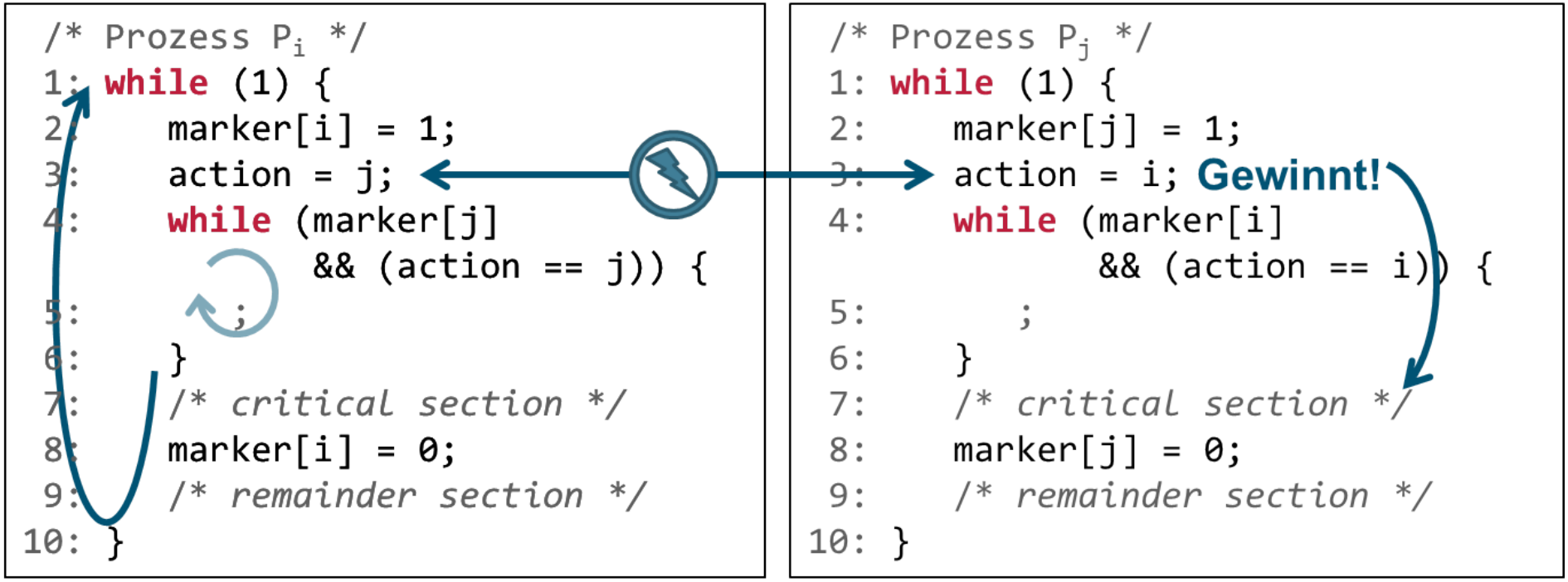
a) Synchronisationsalgorithmen

a) Handelt es sich bei dem Algorithmus um eine valide Lösung zur Koordination eines kritischen Abschnitts zwischen zwei Prozessen?

- Ja. Idee stammt von G. L. Peterson im Jahre 1981
- P_i kann den kritischen Abschnitt nur betreten, wenn P_j nicht eintreten will
  - oder wenn P_j den Vorrang an P_i übergeben hat (durch entsprechendes Setzen von `action`)

a) Synchronisationsalgorithmen

Konfliktfall



b) Synchronisationsalgorithmen

b) Welche Schwachstellen könnte dieser Algorithmus in der Praxis haben?

b) Synchronisationsalgorithmen

b) Welche Schwachstellen könnte dieser Algorithmus in der Praxis haben?

- Aufwand ist relativ hoch:
  - So wie gegeben, ist die Lösung beschränkt auf zwei Prozesse
- Compiler und CPU können Anweisungsreihenfolge ändern
  - Stichwort: *Out-of-Order Execution*

c) Synchronisationsalgorithmen

Ist das Programm richtig synchronisiert?

globale Variable: `int mutex = 1;`

thread_1

```
save_world{
  while( mutex == 0 )
  { /*wait*/ }
  mutex = 0;
  //critical section
  mutex = 1;
}
```

thread_2

```
destroy_world{
  while( mutex == 0 )
  { /*wait*/ }
  mutex = 0;
  //critical section
  mutex = 1;
}
```

c) Synchronisationsalgorithmen

Ist das Programm richtig synchronisiert?

```
/* global */
int mutex = 1;
```

```
safe_world {
  while ( mutex == 0 ) {
    /* wait */
  }
  mutex = 0;
  /* critical section */
  mutex = 1;
}
```

```
destroy_world {
  while ( mutex == 0 ) {
    /* wait */
  }
  mutex = 0;
  /* critical section */
  mutex = 1;
}
```

Antwort:

- Das Programm ist **nicht** richtig synchronisiert.
- Beide Threads können den kritischen Bereich zeitgleich betreten.
- Grund: Die Bedingung in der `while`-Schleife kann in beiden Threads gleichzeitig erfüllt sein.

c) Synchronisationsalgorithmen

Weitere Synchronisationsbeispiele gibt es hier:
`http://deadlockempire.github.io/`

Übersicht

Synchronisation mit Mutex und Semaphoren a)

a) Die folgende Tabelle beschreibt eine nebenläufige Ausführung von zwei Prozessen. Als Ergebnis erhält man eine Folge von [tick] und [tock]....

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B

repeat forever:
  down(baz)
  down(foo)
  print([tick])
  up(bar)
  up(baz)
	repeat 10x:
	  down(baz)
	  down(bar)
	  down(bar)
	  print([tock])
	  up(foo)
	  up(baz)

Synchronisation mit Mutex und Semaphoren a)

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B

repeat forever:
  down(baz)
  down(foo)
  print([tick])
  up(bar)
  up(baz)
	repeat 10x:
	  down(baz)
	  down(bar)
	  down(bar)
	  print([tock])
	  up(foo)
	  up(baz)

1. [tick] [tock] [tick] [tock] [tick] [tock] [tick] [tock] [tick] [tock]
2. [tock] [tick] [tock] [tick] [tock] [tick] [tock] [tick] [tock] [tick]
3. [tick] [tick] [tock] [tick] [tick] [tock] [tick] [tick] [tock] [tick]
4. [tock] [tick] [tick] [tock] [tick] [tick] [tock] [tick] [tick] [tock]

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)
foo -1 bar +1 baz ±0	foo +1 bar -2 baz ±0

Notation: (foo, bar)

1. [tick] [tock] [tick] [tock] [tick] [tock] [tick] [tock] [tick] [tock]

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)
foo -1 bar +1 baz ±0	foo +1 bar -2 baz ±0

Notation: (foo, bar)

1. [tick] [tock] [tick] [tock] [tick] [tock] [tick] [tock] [tick] [tock]

   (2,6)

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)
foo -1 bar +1 baz ±0	foo +1 bar -2 baz ±0

Notation: (foo, bar)

1. [tick] [tock] [tick] [tock] [tick] [tock] [tick] [tock] [tick] [tock]

   (2,6) (3,4)

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)
foo -1 bar +1 baz ±0	foo +1 bar -2 baz ±0

Notation: (foo, bar)

1. [tick] [tock] [tick] [tock] [tick] [tock] [tick] [tock] [tick] [tock]

   (2,6) (3,4) (2,5)

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)
foo -1 bar +1 baz ±0	foo +1 bar -2 baz ±0

Notation: (foo, bar)

1. [tick] [tock] [tick] [tock] [tick] [tock] [tick] [tock] [tick] [tock]

   (2,6) (3,4) (2,5) (3,3)

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)
foo -1 bar +1 baz ±0	foo +1 bar -2 baz ±0

Notation: (foo, bar)

1. [tick] [tock] [tick] [tock] [tick] [tock] [tick] [tock] [tick] [tock]

   (2,6) (3,4) (2,5) (3,3) (2,4)

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)
foo -1 bar +1 baz ±0	foo +1 bar -2 baz ±0

Notation: (foo, bar)

1. [tick] [tock] [tick] [tock] [tick] [tock] [tick] [tock] [tick] [tock]

   (2,6) (3,4) (2,5) (3,3) (2,4) (3,2)

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)
foo -1 bar +1 baz ±0	foo +1 bar -2 baz ±0

Notation: (foo, bar)

1. [tick] [tock] [tick] [tock] [tick] [tock] [tick] [tock] [tick] [tock]

(2,6) (3,4) (2,5) (3,3) (2,4) (3,2) (2,3)

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)
foo -1 bar +1 baz ±0	foo +1 bar -2 baz ±0

Notation: (foo, bar)

1. [tick] [tock] [tick] [tock] [tick] [tock] [tick] [tock] [tick] [tock]

   (2,6) (3,4) (2,5) (3,3) (2,4) (3,2) (2,3) (3,1)

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)
foo -1 bar +1 baz ±0	foo +1 bar -2 baz ±0

Notation: (foo, bar)

1. [tick] [tock] [tick] [tock] [tick] [tock] [tick] [tock] [tick] [tock]

   (2,6) (3,4) (2,5) (3,3) (2,4) (3,2) (2,3) (3,1) (2,2)

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)
foo -1 bar +1 baz ±0	foo +1 bar -2 baz ±0

Notation: (foo, bar)

1.	[tick]	[tock]	[tick]	[tock]	[tick]	[tock]	[tick]	[tock]	[tick]	[tock]
	(2,6)	(3,4)	(2,5)	(3,3)	(2,4)	(3,2)	(2,3)	(3,1)	(2,2)	(3,0)

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)
foo -1 bar +1 baz ±0	foo +1 bar -2 baz ±0

Notation: (foo, bar)

1.	[tick]	[tock]	[tick]	[tock]	[tick]	[tock]	[tick]	[tock]	[tick]	[tock]
✓	(2,6)	(3,4)	(2,5)	(3,3)	(2,4)	(3,2)	(2,3)	(3,1)	(2,2)	(3,0)

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)
foo -1 bar +1 baz ±0	foo +1 bar -2 baz ±0

Notation: (foo, bar)

2. [tock] [tick] [tock] [tick] [tock] [tick] [tock] [tick] [tock] [tick]

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)
foo -1 bar +1 baz ±0	foo +1 bar -2 baz ±0

Notation: (foo, bar)

2. [tock] [tick] [tock] [tick] [tock] [tick] [tock] [tick] [tock] [tick]

(4,3)

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)
foo -1 bar +1 baz ±0	foo +1 bar -2 baz ±0

Notation: (foo, bar)

2. [tock] [tick] [tock] [tick] [tock] [tick] [tock] [tick] [tock] [tick]

(4,3) (3,4)

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)
foo -1 bar +1 baz ±0	foo +1 bar -2 baz ±0

Notation: (foo, bar)

2. [tock] [tick] [tock] [tick] [tock] [tick] [tock] [tick] [tock] [tick]

(4,3) (3,4) (4,2)

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)
foo -1 bar +1 baz ±0	foo +1 bar -2 baz ±0

Notation: (foo, bar)

2. [tock] [tick] [tock] [tick] [tock] [tick] [tock] [tick] [tock] [tick]

(4,3) (3,4) (4,2) (3,3)

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)
foo -1 bar +1 baz ±0	foo +1 bar -2 baz ±0

Notation: (foo, bar)

2. [tock] [tick] [tock] [tick] [tock] [tick] [tock] [tick] [tock] [tick]

(4,3) (3,4) (4,2) (3,3) (4,1)

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)
foo -1 bar +1 baz ±0	foo +1 bar -2 baz ±0

Notation: (foo, bar)

2. [tock] [tick] [tock] [tick] [tock] [tick] [tock] [tick] [tock] [tick]

(4,3) (3,4) (4,2) (3,3) (4,1) (3,2)

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)
foo -1 bar +1 baz ±0	foo +1 bar -2 baz ±0

Notation: (foo, bar)

2. [tock] [tick] [tock] [tick] [tock] [tick] [tock] [tick] [tock] [tick]

(4,3) (3,4) (4,2) (3,3) (4,1) (3,2) (4,0)

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)
foo -1 bar +1 baz ±0	foo +1 bar -2 baz ±0

Notation: (foo, bar)

2. [tock] [tick] [tock] [tick] [tock] [tick] [tock] [tick] [tock] [tick]

(4,3) (3,4) (4,2) (3,3) (4,1) (3,2) (4,0) (3,1)

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)

A: foo -1, bar +1, baz ±0

B: foo +1, bar -2, baz ±0

Notation: (foo, bar)

2.	[tock]	[tick]	[tock]	[tick]	[tock]	[tick]	[tock]	[tick]	[tock]	[tick]
X	(4,3)	(3,4)	(4,2)	(3,3)	(4,1)	(3,2)	(4,0)	(3,1)	(4,-1)	

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)

A: foo -1, bar +1, baz ±0

B: foo +1, bar -2, baz ±0

Notation: (foo, bar)

2.	[tock]	[tick]	[tock]	[tick]	[tock]	[tick]	[tock]	[tick]	[tock]	[tick]
X	(4,3)	(3,4)	(4,2)	(3,3)	(4,1)	(3,2)	(4,0)	(3,1)	(4,-1)	(3,0)

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)
foo -1 bar +1 baz ±0	foo +1 bar -2 baz ±0

Notation: (foo, bar)

3. [tick] [tick] [tock] [tick] [tick] [tock] [tick] [tick] [tock] [tick]

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)
foo -1 bar +1 baz ±0	foo +1 bar -2 baz ±0

Notation: (foo, bar)

3. [tick] [tick] [tock] [tick] [tick] [tock] [tick] [tick] [tock] [tick]

(2,6)

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)
foo -1 bar +1 baz ±0	foo +1 bar -2 baz ±0

Notation: (foo, bar)

3. [tick] [tick] [tock] [tick] [tick] [tock] [tick] [tick] [tock] [tick]

(2,6) (1,7)

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)
foo -1 bar +1 baz ±0	foo +1 bar -2 baz ±0

Notation: (foo, bar)

3. [tick] [tick] [tock] [tick] [tick] [tock] [tick] [tick] [tock] [tick]

(2,6) (1,7) (2,5)

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)
foo -1 bar +1 baz ±0	foo +1 bar -2 baz ±0

Notation: (foo, bar)

3. [tick] [tick] [tock] [tick] [tick] [tock] [tick] [tick] [tock] [tick]

(2,6) (1,7) (2,5) (1,6)

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)
foo -1 bar +1 baz ±0	foo +1 bar -2 baz ±0

Notation: (foo, bar)

3. [tick] [tick] [tock] [tick] [tick] [tock] [tick] [tick] [tock] [tick]

(2,6) (1,7) (2,5) (1,6) (0,7)

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)
foo -1 bar +1 baz ±0	foo +1 bar -2 baz ±0

Notation: (foo, bar)

3. [tick] [tick] [tock] [tick] [tick] [tock] [tick] [tick] [tock] [tick]

(2,6) (1,7) (2,5) (1,6) (0,7) (1,5)

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)
foo -1 bar +1 baz ±0	foo +1 bar -2 baz ±0

Notation: (foo, bar)

3. [tick] [tick] [tock] [tick] [tick] [tock] [tick] [tick] [tock] [tick]

(2,6) (1,7) (2,5) (1,6) (0,7) (1,5) (0,6)

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)

A: foo -1, bar +1, baz ±0

B: foo +1, bar -2, baz ±0

Notation: (foo, bar)

3.	[tick]	[tick]	[tock]	[tick]	[tick]	[tock]	[tick]	[tick]	[tock]	[tick]
X	(2,6)	(1,7)	(2,5)	(1,6)	(0,7)	(1,5)	(0,6)	(-1,7)		

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)

A: foo -1, bar +1, baz ±0

B: foo +1, bar -2, baz ±0

Notation: (foo, bar)

3.	[tick]	[tick]	[tock]	[tick]	[tick]	[tock]	[tick]	[tick]	[tock]	[tick]
X	(2,6)	(1,7)	(2,5)	(1,6)	(0,7)	(1,5)	(0,6)	(-1,7)	(0,5)	

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)

A: foo -1, bar +1, baz ±0

B: foo +1, bar -2, baz ±0

Notation: (foo, bar)

3.	[tick]	[tick]	[tock]	[tick]	[tick]	[tock]	[tick]	[tick]	[tock]	[tick]
X	(2,6)	(1,7)	(2,5)	(1,6)	(0,7)	(1,5)	(0,6)	(-1,7)	(0,5)	(-1,6)

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)
foo -1 bar +1 baz ±0	foo +1 bar -2 baz ±0

Notation: (foo, bar)

4. [tock] [tick] [tick] [tock] [tick] [tick] [tock] [tick] [tick] [tock]

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)
foo -1 bar +1 baz ±0	foo +1 bar -2 baz ±0

Notation: (foo, bar)

4. [tock] [tick] [tick] [tock] [tick] [tick] [tock] [tick] [tick] [tock]

(4,3)

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)
foo -1 bar +1 baz ±0	foo +1 bar -2 baz ±0

Notation: (foo, bar)

4. [tock] [tick] [tick] [tock] [tick] [tick] [tock] [tick] [tick] [tock]

(4,3) (3,4)

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)
foo -1 bar +1 baz ±0	foo +1 bar -2 baz ±0

Notation: (foo, bar)

4. [tock] [tick] [tick] [tock] [tick] [tick] [tock] [tick] [tick] [tock]

(4,3) (3,4) (2,5)

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)
foo -1 bar +1 baz ±0	foo +1 bar -2 baz ±0

Notation: (foo, bar)

4. [tock] [tick] [tick] [tock] [tick] [tick] [tock] [tick] [tick] [tock]

(4,3) (3,4) (2,5) (3,3)

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)
foo -1 bar +1 baz ±0	foo +1 bar -2 baz ±0

Notation: (foo, bar)

4. [tock] [tick] [tick] [tock] [tick] [tick] [tock] [tick] [tick] [tock]

(4,3) (3,4) (2,5) (3,3) (2,4)

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)
foo -1 bar +1 baz ±0	foo +1 bar -2 baz ±0

Notation: (foo, bar)

4. [tock] [tick] [tick] [tock] [tick] [tick] [tock] [tick] [tick] [tock]

(4,3) (3,4) (2,5) (3,3) (2,4) (1,5)

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)
foo -1 bar +1 baz ±0	foo +1 bar -2 baz ±0

Notation: (foo, bar)

4. [tock] [tick] [tick] [tock] [tick] [tick] [tock] [tick] [tick] [tock]

(4,3) (3,4) (2,5) (3,3) (2,4) (1,5) (2,3)

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)
foo -1 bar +1 baz ±0	foo +1 bar -2 baz ±0

Notation: (foo, bar)

4. [tock] [tick] [tick] [tock] [tick] [tick] [tock] [tick] [tick] [tock]

(4,3) (3,4) (2,5) (3,3) (2,4) (1,5) (2,3) (1,4)

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)
foo -1 bar +1 baz ±0	foo +1 bar -2 baz ±0

Notation: (foo, bar)

4. [tock] [tick] [tick] [tock] [tick] [tick] [tock] [tick] [tick] [tock]

(4,3) (3,4) (2,5) (3,3) (2,4) (1,5) (2,3) (1,4) (0,5)

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)
foo -1 bar +1 baz ±0	foo +1 bar -2 baz ±0

Notation: (foo, bar)

4.	[tock]	[tick]	[tick]	[tock]	[tick]	[tick]	[tock]	[tick]	[tick]	[tock]
	(4,3)	(3,4)	(2,5)	(3,3)	(2,4)	(1,5)	(2,3)	(1,4)	(0,5)	(1,3)

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)

foo -1, bar +1, baz ±0

foo +1, bar -2, baz ±0

Notation: (foo, bar)

4.	[tock]	[tick]	[tick]	[tock]	[tick]	[tick]	[tock]	[tick]	[tick]	[tock]
✓	(4,3)	(3,4)	(2,5)	(3,3)	(2,4)	(1,5)	(2,3)	(1,4)	(0,5)	(1,3)

a) Synchronisation mit Mutex und Semaphoren

a) ...Welche Ausführungsreihenfolgen sind möglich?

```
Semaphore baz = 1;
Semaphore foo = 3;
Semaphore bar = 5;
```

Prozess A	Prozess B
repeat forever: down(baz) down(foo) print([tick]) up(bar) up(baz)	repeat 10x: down(baz) down(bar) down(bar) print([tock]) up(foo) up(baz)
foo -1 bar +1 baz ±0	foo +1 bar -2 baz ±0

1. [tick] [tock] [tick] [tock] [tick] [tock] [tick] [tock] [tick] [tock] ✓
2. [tock] [tick] [tock] [tick] [tock] [tick] [tock] [tick] [tock] [tick] X
3. [tick] [tick] [tock] [tick] [tick] [tock] [tick] [tick] [tock] [tick] X
4. [tock] [tick] [tick] [tock] [tick] [tick] [tock] [tick] [tick] [tock] ✓

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

1. `semaphore sx=1; semaphore sy=0; semaphore sz=0;`
2. `semaphore sx=0; semaphore sy=1; semaphore sz=0;`
3. `semaphore sx=0; semaphore sy=1; semaphore sz=1;`

Prozess X	Prozess Y	Prozess Z
down(sx)	down(sy)	down(sz)
print(Fiasco)	print(eCos)	print(RIOT)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)
print(Barrelfish)	print(RTEMS)	print(Contiki)
up(sz)	up(sx)	up(sy)
down(sx)	down(sy)	down(sz)
print(SunOS)	print(QNX)	print(TinyOS)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

1. `semaphore sx=1; semaphore sy=0; semaphore sz=0;`

Prozess X	Prozess Y	Prozess Z
down(sx)	down(sy)	down(sz)
print(Fiasco)	print(eCos)	print(RIOT)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)
print(Barrelfish)	print(RTEMS)	print(Contiki)
up(sz)	up(sx)	up(sy)
down(sx)	down(sy)	down(sz)
print(SunOS)	print(QNX)	print(TinyOS)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

1. `semaphore sx=1; semaphore sy=0; semaphore sz=0;`

Prozess X	Prozess Y	Prozess Z
down(sx)	down(sy)	down(sz)
print(Fiasco)	print(eCos)	print(RIOT)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)
print(Barrelfish)	print(RTEMS)	print(Contiki)
up(sz)	up(sx)	up(sy)
down(sx)	down(sy)	down(sz)
print(SunOS)	print(QNX)	print(TinyOS)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)

Fiasco

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

1. `semaphore sx=1; semaphore sy=0; semaphore sz=0;`

Prozess X	Prozess Y	Prozess Z
down(sx)	down(sy)	down(sz)
print(Fiasco)	print(eCos)	print(RIOT)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)
print(Barrelfish)	print(RTEMS)	print(Contiki)
up(sz)	up(sx)	up(sy)
down(sx)	down(sy)	down(sz)
print(SunOS)	print(QNX)	print(TinyOS)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)

Fiasco eCos

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

1. `semaphore sx=1; semaphore sy=0; semaphore sz=0;`

Prozess X	Prozess Y	Prozess Z
down(sx)	down(sy)	down(sz)
print(Fiasco)	print(eCos)	print(RIOT)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)
print(Barrelfish)	print(RTEMS)	print(Contiki)
up(sz)	up(sx)	up(sy)
down(sx)	down(sy)	down(sz)
print(SunOS)	print(QNX)	print(TinyOS)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)

Fiasco eCos

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

1. `semaphore sx=1; semaphore sy=0; semaphore sz=0;`

Prozess X	Prozess Y	Prozess Z
down(sx)	down(sy)	down(sz)
print(Fiasco)	print(eCos)	print(RIOT)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)
print(Barrelfish)	print(RTEMS)	print(Contiki)
up(sz)	up(sx)	up(sy)
down(sx)	down(sy)	down(sz)
print(SunOS)	print(QNX)	print(TinyOS)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)

Fiasco eCos RIOT

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

1. `semaphore sx=1; semaphore sy=0; semaphore sz=0;`

Prozess X	Prozess Y	Prozess Z
down(sx)	down(sy)	down(sz)
print(Fiasco)	print(eCos)	print(RIOT)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)
print(Barrelfish)	print(RTEMS)	print(Contiki)
up(sz)	up(sx)	up(sy)
down(sx)	down(sy)	down(sz)
print(SunOS)	print(QNX)	print(TinyOS)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)

Fiasco eCos RIOT

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

1. `semaphore sx=1; semaphore sy=0; semaphore sz=0;`

Prozess X	Prozess Y	Prozess Z
down(sx)	down(sy)	down(sz)
print(Fiasco)	print(eCos)	print(RIOT)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)
print(Barrelfish)	print(RTEMS)	print(Contiki)
up(sz)	up(sx)	up(sy)
down(sx)	down(sy)	down(sz)
print(SunOS)	print(QNX)	print(TinyOS)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)

Fiasco eCos RIOT Barrelfish

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

1. `semaphore sx=1; semaphore sy=0; semaphore sz=0;`

Prozess X	Prozess Y	Prozess Z
down(sx)	down(sy)	down(sz)
print(Fiasco)	print(eCos)	print(RIOT)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)
print(Barrelfish)	print(RTEMS)	print(Contiki)
up(sz)	up(sx)	up(sy)
down(sx)	down(sy)	down(sz)
print(SunOS)	print(QNX)	print(TinyOS)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)

Fiasco eCos RIOT Barrelfish

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

1. `semaphore sx=1; semaphore sy=0; semaphore sz=0;`

Prozess X	Prozess Y	Prozess Z
down(sx)	down(sy)	down(sz)
print(Fiasco)	print(eCos)	print(RIOT)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)
print(Barrelfish)	print(RTEMS)	print(Contiki)
up(sz)	up(sx)	up(sy)
down(sx)	down(sy)	down(sz)
print(SunOS)	print(QNX)	print(TinyOS)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)

Fiasco eCos RIOT Barrelfish Contiki

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

1. `semaphore sx=1; semaphore sy=0; semaphore sz=0;`

Prozess X	Prozess Y	Prozess Z
down(sx)	down(sy)	down(sz)
print(Fiasco)	print(eCos)	print(RIOT)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)
print(Barrelfish)	print(RTEMS)	print(Contiki)
up(sz)	up(sx)	up(sy)
down(sx)	down(sy)	down(sz)
print(SunOS)	print(QNX)	print(TinyOS)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)

Fiasco eCos RIOT Barrelfish Contiki

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

1. `semaphore sx=1; semaphore sy=0; semaphore sz=0;`

Prozess X	Prozess Y	Prozess Z
down(sx)	down(sy)	down(sz)
print(Fiasco)	print(eCos)	print(RIOT)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)
print(Barrelfish)	print(RTEMS)	print(Contiki)
up(sz)	up(sx)	up(sy)
down(sx)	down(sy)	down(sz)
print(SunOS)	print(QNX)	print(TinyOS)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)

Fiasco eCos RIOT Barrelfish Contiki RTEMS

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

1. `semaphore sx=1; semaphore sy=0; semaphore sz=0;`

Prozess X	Prozess Y	Prozess Z
down(sx)	down(sy)	down(sz)
print(Fiasco)	print(eCos)	print(RIOT)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)
print(Barrelfish)	print(RTEMS)	print(Contiki)
up(sz)	up(sx)	up(sy)
down(sx)	down(sy)	down(sz)
print(SunOS)	print(QNX)	print(TinyOS)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)

Fiasco eCos RIOT Barrelfish Contiki RTEMS

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

1. `semaphore sx=1; semaphore sy=0; semaphore sz=0;`

Prozess X	Prozess Y	Prozess Z
down(sx)	down(sy)	down(sz)
print(Fiasco)	print(eCos)	print(RIOT)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)
print(Barrelfish)	print(RTEMS)	print(Contiki)
up(sz)	up(sx)	up(sy)
down(sx)	down(sy)	down(sz)
print(SunOS)	print(QNX)	print(TinyOS)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)

Fiasco eCos RIOT Barrelfish Contiki RTEMS SunOS

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

1. `semaphore sx=1; semaphore sy=0; semaphore sz=0;`

Prozess X	Prozess Y	Prozess Z
down(sx)	down(sy)	down(sz)
print(Fiasco)	print(eCos)	print(RIOT)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)
print(Barrelfish)	print(RTEMS)	print(Contiki)
up(sz)	up(sx)	up(sy)
down(sx)	down(sy)	down(sz)
print(SunOS)	print(QNX)	print(TinyOS)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)

Fiasco eCos RIOT Barrelfish Contiki RTEMS SunOS

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

1. `semaphore sx=1; semaphore sy=0; semaphore sz=0;`

Prozess X	Prozess Y	Prozess Z
down(sx)	down(sy)	down(sz)
print(Fiasco)	print(eCos)	print(RIOT)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)
print(Barrelfish)	print(RTEMS)	print(Contiki)
up(sz)	up(sx)	up(sy)
down(sx)	down(sy)	down(sz)
print(SunOS)	print(QNX)	print(TinyOS)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)

Fiasco eCos RIOT Barrelfish Contiki RTEMS SunOS QNX

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

1. `semaphore sx=1; semaphore sy=0; semaphore sz=0;`

Prozess X	Prozess Y	Prozess Z
down(sx)	down(sy)	down(sz)
print(Fiasco)	print(eCos)	print(RIOT)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)
print(Barrelfish)	print(RTEMS)	print(Contiki)
up(sz)	up(sx)	up(sy)
down(sx)	down(sy)	down(sz)
print(SunOS)	print(QNX)	print(TinyOS)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)

Fiasco eCos RIOT Barrelfish Contiki RTEMS SunOS QNX

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

1. `semaphore sx=1; semaphore sy=0; semaphore sz=0;`

Prozess X	Prozess Y	Prozess Z
down(sx)	down(sy)	down(sz)
print(Fiasco)	print(eCos)	print(RIOT)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)
print(Barrelfish)	print(RTEMS)	print(Contiki)
up(sz)	up(sx)	up(sy)
down(sx)	down(sy)	down(sz)
print(SunOS)	print(QNX)	print(TinyOS)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)

Fiasco eCos RIOT Barrelfish Contiki RTEMS SunOS QNX TinyOS

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

1. `semaphore sx=1; semaphore sy=0; semaphore sz=0;`

Prozess X	Prozess Y	Prozess Z
down(sx)	down(sy)	down(sz)
print(Fiasco)	print(eCos)	print(RIOT)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)
print(Barrelfish)	print(RTEMS)	print(Contiki)
up(sz)	up(sx)	up(sy)
down(sx)	down(sy)	down(sz)
print(SunOS)	print(QNX)	print(TinyOS)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)

Fiasco eCos RIOT Barrelfish Contiki RTEMS SunOS QNX TinyOS

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

1. `semaphore sx=1; semaphore sy=0; semaphore sz=0;`

Prozess X	Prozess Y	Prozess Z
down(sx)	down(sy)	down(sz)
print(Fiasco)	print(eCos)	print(RIOT)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)
print(Barrelfish)	print(RTEMS)	print(Contiki)
up(sz)	up(sx)	up(sy)
down(sx)	down(sy)	down(sz)
print(SunOS)	print(QNX)	print(TinyOS)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)

Fiasco eCos RIOT Barrelfish Contiki RTEMS SunOS QNX TinyOS

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

2. `semaphore sx=0; semaphore sy=1; semaphore sz=0;`

Prozess X	Prozess Y	Prozess Z
down(sx)	down(sy)	down(sz)
print(Fiasco)	print(eCos)	print(RIOT)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)
print(Barrelfish)	print(RTEMS)	print(Contiki)
up(sz)	up(sx)	up(sy)
down(sx)	down(sy)	down(sz)
print(SunOS)	print(QNX)	print(TinyOS)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

2. `semaphore sx=0; semaphore sy=1; semaphore sz=0;`

Prozess X	Prozess Y	Prozess Z
down(sx)	down(sy)	down(sz)
print(Fiasco)	print(eCos)	print(RIOT)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)
print(Barrelfish)	print(RTEMS)	print(Contiki)
up(sz)	up(sx)	up(sy)
down(sx)	down(sy)	down(sz)
print(SunOS)	print(QNX)	print(TinyOS)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)

eCos

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

2. `semaphore sx=0; semaphore sy=1; semaphore sz=0;`

Prozess X	Prozess Y	Prozess Z
down(sx)	down(sy)	down(sz)
print(Fiasco)	print(eCos)	print(RIOT)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)
print(Barrelfish)	print(RTEMS)	print(Contiki)
up(sz)	up(sx)	up(sy)
down(sx)	down(sy)	down(sz)
print(SunOS)	print(QNX)	print(TinyOS)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)

eCos

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

2. `semaphore sx=0; semaphore sy=1; semaphore sz=0;`

Prozess X	Prozess Y	Prozess Z
down(sx)	down(sy)	down(sz)
print(Fiasco)	print(eCos)	print(RIOT)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)
print(Barrelfish)	print(RTEMS)	print(Contiki)
up(sz)	up(sx)	up(sy)
down(sx)	down(sy)	down(sz)
print(SunOS)	print(QNX)	print(TinyOS)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)

eCos RIOT

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

2. `semaphore sx=0; semaphore sy=1; semaphore sz=0;`

Prozess X	Prozess Y	Prozess Z
down(sx)	down(sy)	down(sz)
print(Fiasco)	print(eCos)	print(RIOT)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)
print(Barrelfish)	print(RTEMS)	print(Contiki)
up(sz)	up(sx)	up(sy)
down(sx)	down(sy)	down(sz)
print(SunOS)	print(QNX)	print(TinyOS)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)

eCos RIOT

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

2. `semaphore sx=0; semaphore sy=1; semaphore sz=0;`

Prozess X	Prozess Y	Prozess Z
down(sx)	down(sy)	down(sz)
print(Fiasco)	print(eCos)	print(RIOT)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)
print(Barrelfish)	print(RTEMS)	print(Contiki)
up(sz)	up(sx)	up(sy)
down(sx)	down(sy)	down(sz)
print(SunOS)	print(QNX)	print(TinyOS)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)

eCos RIOT Fiasco

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

2. `semaphore sx=0; semaphore sy=1; semaphore sz=0;`

Prozess X	Prozess Y	Prozess Z
down(sx)	down(sy)	down(sz)
print(Fiasco)	print(eCos)	print(RIOT)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)
print(Barrelfish)	print(RTEMS)	print(Contiki)
up(sz)	up(sx)	up(sy)
down(sx)	down(sy)	down(sz)
print(SunOS)	print(QNX)	print(TinyOS)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)

eCos RIOT Fiasco

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

2. `semaphore sx=0; semaphore sy=1; semaphore sz=0;`

Prozess X	Prozess Y	Prozess Z
down(sx)	down(sy)	down(sz)
print(Fiasco)	print(eCos)	print(RIOT)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)
print(Barrelfish)	print(RTEMS)	print(Contiki)
up(sz)	up(sx)	up(sy)
down(sx)	down(sy)	down(sz)
print(SunOS)	print(QNX)	print(TinyOS)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)

eCos RIOT Fiasco RTEMS

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

2. `semaphore sx=0; semaphore sy=1; semaphore sz=0;`

Prozess X	Prozess Y	Prozess Z
down(sx)	down(sy)	down(sz)
print(Fiasco)	print(eCos)	print(RIOT)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)
print(Barrelfish)	print(RTEMS)	print(Contiki)
up(sz)	up(sx)	up(sy)
down(sx)	down(sy)	down(sz)
print(SunOS)	print(QNX)	print(TinyOS)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)

eCos RIOT Fiasco RTEMS

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

2. `semaphore sx=0; semaphore sy=1; semaphore sz=0;`

Prozess X	Prozess Y	Prozess Z
down(sx)	down(sy)	down(sz)
print(Fiasco)	print(eCos)	print(RIOT)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)
print(Barrelfish)	print(RTEMS)	print(Contiki)
up(sz)	up(sx)	up(sy)
down(sx)	down(sy)	down(sz)
print(SunOS)	print(QNX)	print(TinyOS)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)

eCos RIOT Fiasco RTEMS Barrelfish

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

2. `semaphore sx=0; semaphore sy=1; semaphore sz=0;`

Prozess X	Prozess Y	Prozess Z
down(sx)	down(sy)	down(sz)
print(Fiasco)	print(eCos)	print(RIOT)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)
print(Barrelfish)	print(RTEMS)	print(Contiki)
up(sz)	up(sx)	up(sy)
down(sx)	down(sy)	down(sz)
print(SunOS)	print(QNX)	print(TinyOS)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)

eCos RIOT Fiasco RTEMS Barrelfish

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

2. `semaphore sx=0; semaphore sy=1; semaphore sz=0;`

Prozess X	Prozess Y	Prozess Z
down(sx)	down(sy)	down(sz)
print(Fiasco)	print(eCos)	print(RIOT)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)
print(Barrelfish)	print(RTEMS)	print(Contiki)
up(sz)	up(sx)	up(sy)
down(sx)	down(sy)	down(sz)
print(SunOS)	print(QNX)	print(TinyOS)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)

eCos RIOT Fiasco RTEMS Barrelfish Contiki

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

2. `semaphore sx=0; semaphore sy=1; semaphore sz=0;`

Prozess X	Prozess Y	Prozess Z
down(sx)	down(sy)	down(sz)
print(Fiasco)	print(eCos)	print(RIOT)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)
print(Barrelfish)	print(RTEMS)	print(Contiki)
up(sz)	up(sx)	up(sy)
down(sx)	down(sy)	down(sz)
print(SunOS)	print(QNX)	print(TinyOS)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)

eCos RIOT Fiasco RTEMS Barrelfish Contiki

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

2. `semaphore sx=0; semaphore sy=1; semaphore sz=0;`

Prozess X	Prozess Y	Prozess Z
down(sx)	down(sy)	down(sz)
print(Fiasco)	print(eCos)	print(RIOT)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)
print(Barrelfish)	print(RTEMS)	print(Contiki)
up(sz)	up(sx)	up(sy)
down(sx)	down(sy)	down(sz)
print(SunOS)	print(QNX)	print(TinyOS)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)

eCos RIOT Fiasco RTEMS Barrelfish Contiki QNX

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

2. `semaphore sx=0; semaphore sy=1; semaphore sz=0;`

Prozess X	Prozess Y	Prozess Z
down(sx)	down(sy)	down(sz)
print(Fiasco)	print(eCos)	print(RIOT)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)
print(Barrelfish)	print(RTEMS)	print(Contiki)
up(sz)	up(sx)	up(sy)
down(sx)	down(sy)	down(sz)
print(SunOS)	print(QNX)	print(TinyOS)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)

eCos RIOT Fiasco RTEMS Barrelfish Contiki QNX

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

2. `semaphore sx=0; semaphore sy=1; semaphore sz=0;`

Prozess X	Prozess Y	Prozess Z
down(sx)	down(sy)	down(sz)
print(Fiasco)	print(eCos)	print(RIOT)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)
print(Barrelfish)	print(RTEMS)	print(Contiki)
up(sz)	up(sx)	up(sy)
down(sx)	down(sy)	down(sz)
print(SunOS)	print(QNX)	print(TinyOS)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)

eCos RIOT Fiasco RTEMS Barrelfish Contiki QNX TinyOS

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

2. `semaphore sx=0; semaphore sy=1; semaphore sz=0;`

Prozess X	Prozess Y	Prozess Z
down(sx)	down(sy)	down(sz)
print(Fiasco)	print(eCos)	print(RIOT)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)
print(Barrelfish)	print(RTEMS)	print(Contiki)
up(sz)	up(sx)	up(sy)
down(sx)	down(sy)	down(sz)
print(SunOS)	print(QNX)	print(TinyOS)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)

eCos RIOT Fiasco RTEMS Barrelfish Contiki QNX TinyOS

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

2. `semaphore sx=0; semaphore sy=1; semaphore sz=0;`

Prozess X	Prozess Y	Prozess Z
down(sx)	down(sy)	down(sz)
print(Fiasco)	print(eCos)	print(RIOT)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)
print(Barrelfish)	print(RTEMS)	print(Contiki)
up(sz)	up(sx)	up(sy)
down(sx)	down(sy)	down(sz)
print(SunOS)	print(QNX)	print(TinyOS)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)

eCos RIOT Fiasco RTEMS Barrelfish Contiki QNX TinyOS SunOS

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

2. `semaphore sx=0; semaphore sy=1; semaphore sz=0;`

Prozess X	Prozess Y	Prozess Z
down(sx)	down(sy)	down(sz)
print(Fiasco)	print(eCos)	print(RIOT)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)
print(Barrelfish)	print(RTEMS)	print(Contiki)
up(sz)	up(sx)	up(sy)
down(sx)	down(sy)	down(sz)
print(SunOS)	print(QNX)	print(TinyOS)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)

eCos RIOT Fiasco RTEMS Barrelfish Contiki QNX TinyOS SunOS

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

2. `semaphore sx=0; semaphore sy=1; semaphore sz=0;`

Prozess X	Prozess Y	Prozess Z
down(sx)	down(sy)	down(sz)
print(Fiasco)	print(eCos)	print(RIOT)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)
print(Barrelfish)	print(RTEMS)	print(Contiki)
up(sz)	up(sx)	up(sy)
down(sx)	down(sy)	down(sz)
print(SunOS)	print(QNX)	print(TinyOS)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)

eCos RIOT Fiasco RTEMS Barrelfish Contiki QNX TinyOS SunOS

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

3. `semaphore sx=0; semaphore sy=1; semaphore sz=1;`

Prozess X	Prozess Y	Prozess Z
down(sx)	down(sy)	down(sz)
print(Fiasco)	print(eCos)	print(RIOT)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)
print(Barrelfish)	print(RTEMS)	print(Contiki)
up(sz)	up(sx)	up(sy)
down(sx)	down(sy)	down(sz)
print(SunOS)	print(QNX)	print(TinyOS)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

3. `semaphore sx=0; semaphore sy=1; semaphore sz=1;`

Prozess X	Prozess Y	Prozess Z
down(sx)	down(sy)	down(sz)
print(Fiasco)	print(eCos)	print(RIOT)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)
print(Barrelfish)	print(RTEMS)	print(Contiki)
up(sz)	up(sx)	up(sy)
down(sx)	down(sy)	down(sz)
print(SunOS)	print(QNX)	print(TinyOS)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)

Keine eindeutige Ausführung möglich!

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

3. `semaphore sx=0; semaphore sy=1; semaphore sz=1;`

Prozess X	Prozess Y	Prozess Z
down(sx)	down(sy)	down(sz)
print(Fiasco)	print(eCos)	print(RIOT)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)
print(Barrelfish)	print(RTEMS)	print(Contiki)
up(sz)	up(sx)	up(sy)
down(sx)	down(sy)	down(sz)
print(SunOS)	print(QNX)	print(TinyOS)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)

Keine eindeutige Ausführung möglich!

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

3. `semaphore sx=0; semaphore sy=1; semaphore sz=1;`

Prozess X	Prozess Y	Prozess Z
down(sx)	down(sy)	down(sz)
print(Fiasco)	print(eCos)	print(RIOT)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)
print(Barrelfish)	print(RTEMS)	print(Contiki)
up(sz)	up(sx)	up(sy)
down(sx)	down(sy)	down(sz)
print(SunOS)	print(QNX)	print(TinyOS)
up(sy)	up(sz)	up(sx)
down(sx)	down(sy)	down(sz)

Keine eindeutige Ausführung möglich!

b) Synchronisation mit Mutex und Semaphoren

b) Ermitteln Sie die Ausführungsabfolgen der Prozesse (X, Y, Z) in Abhängigkeit der gegebenen Mutex-Belegung.

Zusammenfassung:

1. `Fiasco eCos RIOT Barrelfish Contiki RTEMS SunOS QNX TinyOS`
2. `eCos RIOT Fiasco RTEMS Barrelfish Contiki QNX TinyOS SunOS`
3. Keine eindeutige Ausführung möglich!

c) Synchronisation mit Mutex und Semaphoren

c) Ergänzen Sie das Programm durch Mutexe...

Prozess A	Prozess B	Prozess C
`print(kein )`	`print(Erste Regel, )`	`print(ihr )`
`print(den )`	`print(verliert )`	`print(über )`
`print(Club.)`	`print(Wort )`	`print(Fight)`

Ausgabe:

c) Synchronisation mit Mutex und Semaphoren

c) Ergänzen Sie das Programm durch Mutexe...

```
mutex ma  //Mutex-Variablen deklarieren
mutex mb
mutex mc
```

Prozess A	Prozess B	Prozess C
down(ma)	down(mb)	down(mc)
print(kein)	print(Erste Regel,)	print(ihr)
print(den)	print(verliert)	print(über)
print(Club.)	print(Wort)	print(Fight)

Ausgabe:

c) Synchronisation mit Mutex und Semaphoren

c) Ergänzen Sie das Programm durch Mutexe...

```
mutex ma=0
mutex mb=1  //Start mit Prozess B
mutex mc=0
```

Prozess A	Prozess B	Prozess C
down(ma)	down(mb)	down(mc)
print(kein)	print(Erste Regel,)	print(ihr)
print(den)	print(verliert)	print(über)
print(Club.)	print(Wort)	print(Fight)

Ausgabe: Erste Regel,

c) Synchronisation mit Mutex und Semaphoren

c) Ergänzen Sie das Programm durch Mutexe...

```
mutex ma=0
mutex mb=1  //Start mit Prozess B
mutex mc=0
```

Prozess A	Prozess B	Prozess C
down(ma)	down(mb)	down(mc)
print(kein)	print(Erste Regel,)	print(ihr)
	up(mc)	
	down(mb)	
print(den)	print(verliert)	print(über)
print(Club.)	print(Wort)	print(Fight)

Ausgabe: Erste Regel,

c) Synchronisation mit Mutex und Semaphoren

c) Ergänzen Sie das Programm durch Mutexe...

```
mutex ma=0
mutex mb=1  //Start mit Prozess B
mutex mc=0
```

Prozess A	Prozess B	Prozess C
down(ma)	down(mb)	down(mc)
print(kein)	print(Erste Regel,)	print(ihr)
	up(mc)	
	down(mb)	
print(den)	print(verliert)	print(über)
print(Club.)	print(Wort)	print(Fight)

Ausgabe: Erste Regel, ihr

c) Synchronisation mit Mutex und Semaphoren

c) Ergänzen Sie das Programm durch Mutexe...

```
mutex ma=0
mutex mb=1  //Start mit Prozess B
mutex mc=0
```

Prozess A	Prozess B	Prozess C
down(ma)	down(mb)	down(mc)
print(kein)	print(Erste Regel,)	print(ihr)
	up(mc)	**up(mb)**
	down(mb)	down(mc)
print(den)	print(verliert)	print(über)
print(Club.)	print(Wort)	print(Fight)

Ausgabe: Erste Regel, ihr

c) Synchronisation mit Mutex und Semaphoren

c) Ergänzen Sie das Programm durch Mutexe...

```
mutex ma=0
mutex mb=1  //Start mit Prozess B
mutex mc=0
```

Prozess A	Prozess B	Prozess C
down(ma)	down(mb)	down(mc)
print(kein)	print(Erste Regel,)	print(ihr)
	up(mc)	up(mb)
	down(mb)	down(mc)
print(den)	print(verliert)	print(über)
print(Club.)	print(Wort)	print(Fight)

Ausgabe: Erste Regel, ihr verliert

c) Synchronisation mit Mutex und Semaphoren

c) Ergänzen Sie das Programm durch Mutexe...

```
mutex ma=0
mutex mb=1  //Start mit Prozess B
mutex mc=0
```

Prozess A	Prozess B	Prozess C
down(ma)	down(mb)	down(mc)
print(kein)	print(Erste Regel,)	print(ihr)
	up(mc)	up(mb)
	down(mb)	down(mc)
print(den)	print(verliert)	print(über)
	up(ma)	
	down(mb)	
print(Club.)	print(Wort)	print(Fight)

Ausgabe: Erste Regel, ihr verliert

c) Synchronisation mit Mutex und Semaphoren

c) Ergänzen Sie das Programm durch Mutexe...

```
mutex ma=0
mutex mb=1  //Start mit Prozess B
mutex mc=0
```

Prozess A	Prozess B	Prozess C
down(ma)	down(mb)	down(mc)
print(kein)	print(Erste Regel,)	print(ihr)
	up(mc)	up(mb)
	down(mb)	down(mc)
print(den)	print(verliert)	print(über)
	up(ma)	
	down(mb)	
print(Club.)	print(Wort)	print(Fight)

Ausgabe: Erste Regel, ihr verliert kein

c) Synchronisation mit Mutex und Semaphoren

c) Ergänzen Sie das Programm durch Mutexe...

```
mutex ma=0
mutex mb=1  //Start mit Prozess B
mutex mc=0
```

Prozess A	Prozess B	Prozess C
down(ma)	down(mb)	down(mc)
print(kein)	print(Erste Regel,)	print(ihr)
up(mb)	up(mc)	up(mb)
down(ma)	down(mb)	down(mc)
print(den)	print(verliert)	print(über)
	up(ma)	
	down(mb)	
print(Club.)	print(Wort)	print(Fight)

Ausgabe: Erste Regel, ihr verliert kein

c) Synchronisation mit Mutex und Semaphoren

c) Ergänzen Sie das Programm durch Mutexe...

```
mutex ma=0
mutex mb=1  //Start mit Prozess B
mutex mc=0
```

Prozess A	Prozess B	Prozess C
down(ma)	down(mb)	down(mc)
print(kein)	print(Erste Regel,)	print(ihr)
up(mb)	up(mc)	up(mb)
down(ma)	down(mb)	down(mc)
print(den)	print(verliert)	print(über)
	up(ma)	
	down(mb)	
print(Club.)	print(Wort)	print(Fight)

Ausgabe: Erste Regel, ihr verliert kein Wort

c) Synchronisation mit Mutex und Semaphoren

c) Ergänzen Sie das Programm durch Mutexe...

```
mutex ma=0
mutex mb=1  //Start mit Prozess B
mutex mc=0
```

Prozess A	Prozess B	Prozess C
down(ma)	down(mb)	down(mc)
print(kein)	print(Erste Regel,)	print(ihr)
up(mb)	up(mc)	up(mb)
down(ma)	down(mb)	down(mc)
print(den)	print(verliert)	print(über)
	up(ma)	
	down(mb)	
print(Club.)	print(Wort)	print(Fight)
	up(mc)	

Ausgabe: Erste Regel, ihr verliert kein Wort

c) Synchronisation mit Mutex und Semaphoren

c) Ergänzen Sie das Programm durch Mutexe...

```
mutex ma=0
mutex mb=1  //Start mit Prozess B
mutex mc=0
```

Prozess A	Prozess B	Prozess C
down(ma)	down(mb)	down(mc)
print(kein)	print(Erste Regel,)	print(ihr)
up(mb)	up(mc)	up(mb)
down(ma)	down(mb)	**down(mc)**
print(den)	print(verliert)	print(über)
	up(ma)	
	down(mb)	
print(Club.)	print(Wort)	print(Fight)
	up(mc)	

Ausgabe: Erste Regel, ihr verliert kein Wort über

c) Synchronisation mit Mutex und Semaphoren

c) Ergänzen Sie das Programm durch Mutexe...

```
mutex ma=0
mutex mb=1  //Start mit Prozess B
mutex mc=0
```

Prozess A	Prozess B	Prozess C
down(ma)	down(mb)	down(mc)
print(kein)	print(Erste Regel,)	print(ihr)
up(mb)	up(mc)	up(mb)
down(ma)	down(mb)	down(mc)
print(den)	print(verliert)	print(über)
	up(ma)	**up(ma)**
	down(mb)	down(mc)
print(Club.)	print(Wort)	print(Fight)
	up(mc)	

Ausgabe: Erste Regel, ihr verliert kein Wort über

c) Synchronisation mit Mutex und Semaphoren

c) Ergänzen Sie das Programm durch Mutexe...

```
mutex ma=0
mutex mb=1  //Start mit Prozess B
mutex mc=0
```

Prozess A	Prozess B	Prozess C
down(ma)	down(mb)	down(mc)
print(kein)	print(Erste Regel,)	print(ihr)
up(mb)	up(mc)	up(mb)
down(ma)	down(mb)	down(mc)
print(den)	print(verliert)	print(über)
	up(ma)	up(ma)
	down(mb)	down(mc)
print(Club.)	print(Wort)	print(Fight)
	up(mc)	

Ausgabe: Erste Regel, ihr verliert kein Wort über den

c) Synchronisation mit Mutex und Semaphoren

c) Ergänzen Sie das Programm durch Mutexe...

```
mutex ma=0
mutex mb=1  //Start mit Prozess B
mutex mc=0
```

Prozess A	Prozess B	Prozess C
down(ma)	down(mb)	down(mc)
print(kein)	print(Erste Regel,)	print(ihr)
up(mb)	up(mc)	up(mb)
down(ma)	down(mb)	down(mc)
print(den)	print(verliert)	print(über)
up(mc)	up(ma)	up(ma)
down(ma)	down(mb)	down(mc)
print(Club.)	print(Wort)	print(Fight)
	up(mc)	

Ausgabe: Erste Regel, ihr verliert kein Wort über den

c) Synchronisation mit Mutex und Semaphoren

c) Ergänzen Sie das Programm durch Mutexe...

```
mutex ma=0
mutex mb=1  //Start mit Prozess B
mutex mc=0
```

Prozess A	Prozess B	Prozess C
down(ma)	down(mb)	down(mc)
print(kein)	print(Erste Regel,)	print(ihr)
up(mb)	up(mc)	up(mb)
down(ma)	down(mb)	down(mc)
print(den)	print(verliert)	print(über)
up(mc)	up(ma)	up(ma)
down(ma)	down(mb)	**down(mc)**
print(Club.)	print(Wort)	print(Fight)
	up(mc)	

Ausgabe: Erste Regel, ihr verliert kein Wort über den Fight

c) Synchronisation mit Mutex und Semaphoren

c) Ergänzen Sie das Programm durch Mutexe...

```
mutex ma=0
mutex mb=1  //Start mit Prozess B
mutex mc=0
```

Prozess A	Prozess B	Prozess C
down(ma)	down(mb)	down(mc)
print(kein)	print(Erste Regel,)	print(ihr)
up(mb)	up(mc)	up(mb)
down(ma)	down(mb)	down(mc)
print(den)	print(verliert)	print(über)
up(mc)	up(ma)	up(ma)
down(ma)	down(mb)	down(mc)
print(Club.)	print(Wort)	print(Fight)
	up(mc)	**up(ma)**

Ausgabe: Erste Regel, ihr verliert kein Wort über den Fight

c) Synchronisation mit Mutex und Semaphoren

c) Ergänzen Sie das Programm durch Mutexe...

```
mutex ma=0
mutex mb=1  //Start mit Prozess B
mutex mc=0
```

Prozess A	Prozess B	Prozess C
down(ma)	down(mb)	down(mc)
print(kein)	print(Erste Regel,)	print(ihr)
up(mb)	up(mc)	up(mb)
down(ma)	down(mb)	down(mc)
print(den)	print(verliert)	print(über)
up(mc)	up(ma)	up(ma)
down(ma)	down(mb)	down(mc)
print(Club.)	print(Wort)	print(Fight)
	up(mc)	up(ma)

Ausgabe: Erste Regel, ihr verliert kein Wort über den Fight Club.

c) Synchronisation mit Mutex und Semaphoren

c) Ergänzen Sie das Programm durch Mutexe...

```
mutex ma=0
mutex mb=1  //Start mit Prozess B
mutex mc=0
```

Prozess A	Prozess B	Prozess C
down(ma)	down(mb)	down(mc)
print(kein)	print(Erste Regel,)	print(ihr)
up(mb)	up(mc)	up(mb)
down(ma)	down(mb)	down(mc)
print(den)	print(verliert)	print(über)
up(mc)	up(ma)	up(ma)
down(ma)	down(mb)	down(mc)
print(Club.)	print(Wort)	print(Fight)
up(mb)	up(mc)	up(ma)

Ausgabe: Erste Regel, ihr verliert kein Wort über den Fight Club.

mb=1;

Übersicht

a) Erzeuger-Verbraucher-Problem

a) Vervollständigen Sie den folgenden Code zu einer korrekten Lösung.

- 1 Erzeuger-Thread
- *n* Verbraucher-Threads
- `int sem_init(sem_t *sem, int pshared, unsigned int value);`
- Increment: `int sem_post(sem_t *sem);`
- Decrement: `int sem_wait(sem_t *sem);`
- Verhalten:
  - Beim Schreiben darf nur **ein** Thread auf Speicher zugreifen
  - Beim Lesen dürfen **beliebig viele** Threads auf Speicher zugreifen
  - Lesen und Schreiben dürfen **nicht** gleichzeitig erfolgen

Erzeuger-Verbraucher-Problem

```
/*Initialisierung*/
int reader_counter = 0; sem_t sem_read, sem_write;
void initialize() {
  sem_init( &sem_read,  0, _ );
  sem_init( &sem_write, 0, _ );
}
```

```
void write_data (blob_t *data){
  ______________________;
  write_to_shared_memory(data); // Exklusiver Zugriff benötigt
  ______________________;
}
```

Erzeuger-Verbraucher-Problem

```
/*Initialisierung*/
int reader_counter = 0; sem_t sem_read, sem_write;
void initialize() {
  sem_init( &sem_read,  0, _ );
  sem_init( &sem_write, 0, _ );
}
```

```
void write_data (blob_t *data){
  sem_wait( &sem_write );        // Nur ein Schreiber erlaubt
  write_to_shared_memory(data); // Exklusiver Zugriff benötigt
  ____________________;
}
```

Erzeuger-Verbraucher-Problem

```
/*Initialisierung*/
int reader_counter = 0; sem_t sem_read, sem_write;
void initialize() {
  sem_init( &sem_read,  0, _ );
  sem_init( &sem_write, 0, _ );
}
```

```
void write_data (blob_t *data){
  sem_wait( &sem_write );         // Nur ein Schreiber erlaubt
  write_to_shared_memory(data); // Exklusiver Zugriff gewährleistet
  sem_post( &sem_write );     // Lesen und Schreiben wieder möglich
}
```

Erzeuger-Verbraucher-Problem

```
/*Initialisierung*/
int reader_counter = 0; sem_t sem_read, sem_write;
void initialize() {
  sem_init( &sem_read,  0, _ );
  sem_init( &sem_write, 0, 1 ); //Initial das Schreiben erlauben
}
```

```
void write_data (blob_t *data){
  sem_wait( &sem_write );         // Nur ein Schreiber erlaubt
  write_to_shared_memory(data); // Exklusiver Zugriff gewährleistet
  sem_post( &sem_write );     // Lesen und Schreiben wieder möglich
}
```

Erzeuger-Verbraucher-Problem

```
/*Initialisierung*/
int reader_counter = 0; sem_t sem_read, sem_write;
void initialize() {
  sem_init( &sem_read,  0, _ );
  sem_init( &sem_write, 0, 1 ); //Initial das Schreiben erlauben
}
```

```
void read_data(blob_t *buffer){
  ______________________;
  reader_counter++;
  if(reader_counter == 1){_____________________;}
  ______________________;
  read_from_shared_memory(buffer); // exklusiver Lese-Zugriff
  ______________________;
  reader_counter--;
  if(reader_counter == 0){_____________________;}
  ______________________;
}
```

Erzeuger-Verbraucher-Problem

```
/*Initialisierung*/
int reader_counter = 0; sem_t sem_read, sem_write;
void initialize() {
  sem_init( &sem_read,  0, _ );
  sem_init( &sem_write, 0, 1 ); //Initial das Schreiben erlauben
}
```

```
void read_data(blob_t *buffer){
  ______________________;
  reader_counter++;
  if(reader_counter == 1){_____________________;}
  sem_wait( &sem_read ); // FALSCHE LÖSUNG
  read_from_shared_memory(buffer); // exklusiver Lese-Zugriff
  sem_post( &sem_read ); // FALSCHE LÖSUNG
  reader_counter--;
  if(reader_counter == 0){_____________________;}
  ______________________;
} // GLEICHZEITIG LESEN UND SCHREIBEN MÖGLICH
```

Erzeuger-Verbraucher-Problem

```
/*Initialisierung*/
int reader_counter = 0; sem_t sem_read, sem_write;
void initialize() {
  sem_init( &sem_read,  0, _ );
  sem_init( &sem_write, 0, 1 ); //Initial das Schreiben erlauben
}
```

```
void read_data(blob_t *buffer){
  sem_wait( &sem_write ); // FALSCHE LÖSUNG
  reader_counter++;
  if(reader_counter == 1){______________________;}
  ______________________;
  read_from_shared_memory(buffer); // exklusiver Lese-Zugriff
  ______________________;
  reader_counter--;
  if(reader_counter == 0){______________________;}
  sem_post( &sem_write ); // FALSCHE LÖSUNG
}// MAXIMAL EIN LESER MÖGLICH
```

Erzeuger-Verbraucher-Problem

```
/*Initialisierung*/
int reader_counter = 0; sem_t sem_read, sem_write;
void initialize() {
  sem_init( &sem_read,  0, _ );
  sem_init( &sem_write, 0, 1 ); //Initial das Schreiben erlauben
}
```

```
void read_data(blob_t *buffer){
  ______________________;
  reader_counter++;
  if(reader_counter == 1){sem_wait( &sem_write );} //kein Schreiben
  ______________________;
  read_from_shared_memory(buffer); // exklusiver Lese-Zugriff
  ______________________;
  reader_counter--;
  if(reader_counter == 0){sem_post( &sem_write );}//Schreibfreigabe
  ______________________;
}
```

Erzeuger-Verbraucher-Problem

```
/*Initialisierung*/
int reader_counter = 0; sem_t sem_read, sem_write;
void initialize() {
  sem_init( &sem_read,  0, _ );
  sem_init( &sem_write, 0, 1 ); //Initial das Schreiben erlauben
}
```

```
void read_data(blob_t *buffer){
  ______________________;
  reader_counter++; //Darf nur von einem Leser verändert werden
  if(reader_counter == 1){sem_wait( &sem_write );} //kein Schreiben
  ______________________;
  read_from_shared_memory(buffer); // exklusiver Lese-Zugriff
  ______________________;
  reader_counter--; //Darf nur von einem Leser verändert werden
  if(reader_counter == 0){sem_post( &sem_write );}//Schreibfreigabe
  ______________________;
}
```

Erzeuger-Verbraucher-Problem

```
/*Initialisierung*/
int reader_counter = 0; sem_t sem_read, sem_write;
void initialize() {
  sem_init( &sem_read,  0, _ );
  sem_init( &sem_write, 0, 1 ); //Initial das Schreiben erlauben
}
```

```
void read_data(blob_t *buffer){
  sem_wait( &sem_read );
  reader_counter++; //kann nur von einem Leser verändert werden
  if(reader_counter == 1){sem_wait( &sem_write );} //kein Schreiben
  sem_post( &sem_read );
  read_from_shared_memory(buffer); // exklusiver Lese-Zugriff
  ______________________;
  reader_counter--; //Darf nur von einem Leser verändert werden
  if(reader_counter == 0){sem_post( &sem_write );}//Schreibfreigabe
  ______________________;
}
```

Erzeuger-Verbraucher-Problem

```
/*Initialisierung*/
int reader_counter = 0; sem_t sem_read, sem_write;
void initialize() {
  sem_init( &sem_read,  0, _ );
  sem_init( &sem_write, 0, 1 ); //Initial das Schreiben erlauben
}
```

```
void read_data(blob_t *buffer){
  sem_wait( &sem_read );
  reader_counter++; //kann nur von einem Leser verändert werden
  if(reader_counter == 1){sem_wait( &sem_write );} //kein Schreiben
  sem_post( &sem_read );
  read_from_shared_memory(buffer); // exklusiver Lese-Zugriff
  sem_wait( &sem_read );
  reader_counter--; //kann nur von einem Leser verändert werden
  if(reader_counter == 0){sem_post( &sem_write );}//Schreibfreigabe
  sem_post( &sem_read );
}
```

Erzeuger-Verbraucher-Problem

```
/*Initialisierung*/
int reader_counter = 0; sem_t sem_read, sem_write;
void initialize() {
  sem_init( &sem_read,  0, 1 ); //Initial counter-Zugriff erlauben
  sem_init( &sem_write, 0, 1 ); //Initial das Schreiben erlauben
}
```

```
void read_data(blob_t *buffer){
  sem_wait( &sem_read );
  reader_counter++; //kann nur von einem Leser verändert werden
  if(reader_counter == 1){sem_wait( &sem_write );} //kein Schreiben
  sem_post( &sem_read );
  read_from_shared_memory(buffer); // exklusiver Lese-Zugriff
  sem_wait( &sem_read );
  reader_counter--; //kann nur von einem Leser verändert werden
  if(reader_counter == 0){sem_post( &sem_write );}//Schreibfreigabe
  sem_post( &sem_read );
}
```

b) Starvation

b) Können Leser oder Schreiber im oben beschriebenen Algorithmus verhungern?

b) Starvation

b) Können Leser oder Schreiber im oben beschriebenen Algorithmus verhungern?

Ja, denn solange mindestens ein Leser auf den Speicher zugreift, wird `sem_write()` nicht freigegeben!

→ Der Schreiber bekommt dadurch keinen Zugriff auf den Speicher