3.6 Kooperation und Koordination

- Beispiel: Beobachter und Protokollierer

  Mittels Induktionsschleife werden Fahrzeuge gezählt. Protokollierung der Anzahl vorbeigekommener Fahrzeuge im Abstand von 10 Minuten.

```
x = 0;
```

Beobachter

```
on indication
    x++;
```

Protokollierer

```
every 10 minutes
    y = x;
    x = 0;
    cout << ... << y << ... ;
```

Gemeinsame Daten

```
shared_int counter = 0;
```

Beobachter

```
int observer()
{ char c;
  while (TRUE) {
    cin.get(c);   // Simulation eines Impulses
                  // der Induktionsschleife durch
                  // drücken einer Terminal-Taste
    counter++;
  }
}
```

Protokollierer

```
int recorder()
{ int last_count;
  cout << "\n\n";
  while (TRUE) {
    sleep(5);   // 5 Sekunden pausieren
    last_count = counter;
    sleep(1);   // 1 Sekunde pausieren
    counter = 0;
    cout << "Anzahl beobachteter Ereignisse: "
         << last_count << "\n" << flush;
  }
}
```

Beobachtetes Ergebnis:

```
Anzahl beobachteter Ereignisse: 0
Anzahl beobachteter Ereignisse: 0
123456789
Anzahl beobachteter Ereignisse: 10
Anzahl beobachteter Ereignisse: 0
Anzahl beobachteter Ereignisse: 0
123456789
Anzahl beobachteter Ereignisse: 10
123456789
Anzahl beobachteter Ereignisse: 1
Anzahl beobachteter Ereignisse: 0
Anzahl beobachteter Ereignisse: 0
Anzahl beobachteter Ereignisse: 0
123456789
Anzahl beobachteter Ereignisse: 0
Anzahl beobachteter Ereignisse: 0
Anzahl beobachteter Ereignisse: 0
123456789
Anzahl beobachteter Ereignisse: 10
Anzahl beobachteter Ereignisse: 0
Anzahl beobachteter Ereignisse: 0
123456789
Anzahl beobachteter Ereignisse: 10
Anzahl beobachteter Ereignisse: 0
123456789
Anzahl beobachteter Ereignisse: 5
Anzahl beobachteter Ereignisse: 0
Anzahl beobachteter Ereignisse: 0
123456789
Anzahl beobachteter Ereignisse: 4
Anzahl beobachteter Ereignisse: 9
Anzahl beobachteter Ereignisse: 0
Anzahl beobachteter Ereignisse: 0
```

Ursache: Konkurrierende Ausführung der Anweisungen

```
counter++;
```

und

```
counter = 0;
```

T_0: Beobachter	$Register_B$ = counter	($Register_B$ = 8)
T_1: Beobachter	$Register_B$ = $Register_B$ + 1	($Register_B$ = 9)
T_2: Protokollierer	$Register_P$ = 0	($Register_P$ = 0)
T_3: Protokollierer	counter = $Register_P$	(counter = 0)
T_4: Beobachter	counter = $Register_B$	(counter = 9 !!)

d. h. das Rücksetzen des Zählers geht verloren, also werden Impulse mehrfach gezählt.

T_0: Protokollierer	$Register_P$ = counter	($Register_P$ = 6)
T_1: Protokollierer	last_counter = $Register_P$	(last_counter = 6)
T_2: Protokollierer	$Register_P$ = 0	($Register_P$ = 0)
T_3: Beobachter	$Register_B$ = counter	($Register_B$ = 6)
T_4: Beobachter	$Register_B$ = $Register_B$+1	($Register_B$ = 7)
T_5: Beobachter	counter = $Register_B$	(counter = 7)
T_6: Protokollierer	counter = $Register_P$	(counter = 0)

d. h. ein Impuls geht bei der Zählung verloren.

Abhilfe: Manipulation gemeinsamer Daten nur unter gegenseitigem Ausschluß.

- Abschnitte, in denen gemeinsame Daten bearbeitet werden heißen kritische Abschnitte.
- Die Anweisungen kritischer Abschnitte müssen atomar ausgeführt werden, d. h. ihre Abarbeitung erscheint für andere Aktivitätsträger als zeitlich unteilbar.

Problem: Wie kann der gegenseitige Ausschluß erzielt werden?

Nachfolgende Problemanalyse folgt den Arbeiten von Dijkstra.

1. Lösungsversuch

- Initialisierung

```
shared_int turn = 1;
```

- Aktivitätsträger 0

```
int thread_0()
{ shared_int turn;
  while (TRUE) {
    while (turn == 1) ;
    critical_section_0();
    turn = 1;
    remainder_0();
  }
}
```

- Aktivitätsträger 1

```
int thread_1()
{ shared_int turn;
  while (TRUE) {
    while (turn == 0) ;
    critical_section_1();
    turn = 0;
    remainder_1();
  }
}
```

Problem: Läßt nur alternierende Abarbeitung zu (unvollständige Implementation)

2. Lösungsversuch

- Initialisierung

```
shared_int thread_0_in_crit = FALSE,
           thread_1_in_crit = FALSE;
```

- Aktivitätsträger 0

```
int thread_0()
{ shared_int thread_0_in_crit, thread_1_in_crit;
  while (TRUE) {
    while (thread_1_in_crit) ;    // active waiting
    thread_0_in_crit = TRUE;
    critical_section_0();
    thread_0_in_crit = FALSE;
    remainder_0();
  }
}
```

- Aktivitätsträger 1

```
int thread_1()
{ shared_int thread_0_in_crit, thread_1_in_crit;
  while (TRUE) {
    while (thread_0_in_crit) ;
    thread_1_in_crit = TRUE;
    critical_section_1();
    thread_1_in_crit = FALSE;
    remainder_1();
  }
}
```

Problem: Gegenseitiger Ausschluß wird nicht mit Sicherheit erreicht (Implementation unsicher);
aktives Warten (Implementation nicht prompt)

3. Versuch

- Initialisierung

```
shared_int thread_0_in_crit = FALSE,
           thread_1_in_crit = FALSE;
```

- Aktivitätsträger 0

```
int thread_0()
{ shared_int thread_0_in_crit, thread_1_in_crit;
  while (TRUE) {
    while (!thread_0_in_crit) {
      thread_0_in_crit = TRUE;
      if (thread_1_in_crit)
        { thread_0_in_crit = FALSE; }
    }
    critical_section_0();
    thread_0_in_crit = FALSE;
    remainder_0();
  }
}
```

- Aktivitätsträger 1

```
int thread_1()
{ shared_int thread_0_in_crit, thread_1_in_crit;
  while (TRUE) {
    while (!thread_1_in_crit) {
      thread_1_in_crit = TRUE;
      if (thread_0_in_crit)
        { thread_1_in_crit = FALSE; }
    }
    critical_section_1();
    thread_1_in_crit = FALSE;
    remainder_1();
  }
}
```

Problem: Überhöflich (Bitte nach Ihnen!, starvation)

Algorithmus von Peterson (1981)

- Initialisierung

```
shared_int turn = 0,
           interested[2] = {FALSE, FALSE};
```

- Aktivitätsträger 0

```
thread_0()
{
  while (TRUE) {
    interested[0] = TRUE;
    turn = 0;
    while (turn == 0 && interested[1] == TRUE) ;
    critical_section_0();
    interested[0] = FALSE;
    remainder_0();
  }
}
```

- Aktivitätsträger 1

```
thread_1()
{
  while (TRUE) {
    interested[1] = TRUE;
    turn = 1;
    while (turn == 1 && interested[0] == TRUE) ;
    critical_section_1();
    interested[1] = FALSE;
    remainder_1();
  }
}
```

Problem: Aktives Warten

- Aktivitätsträger 0

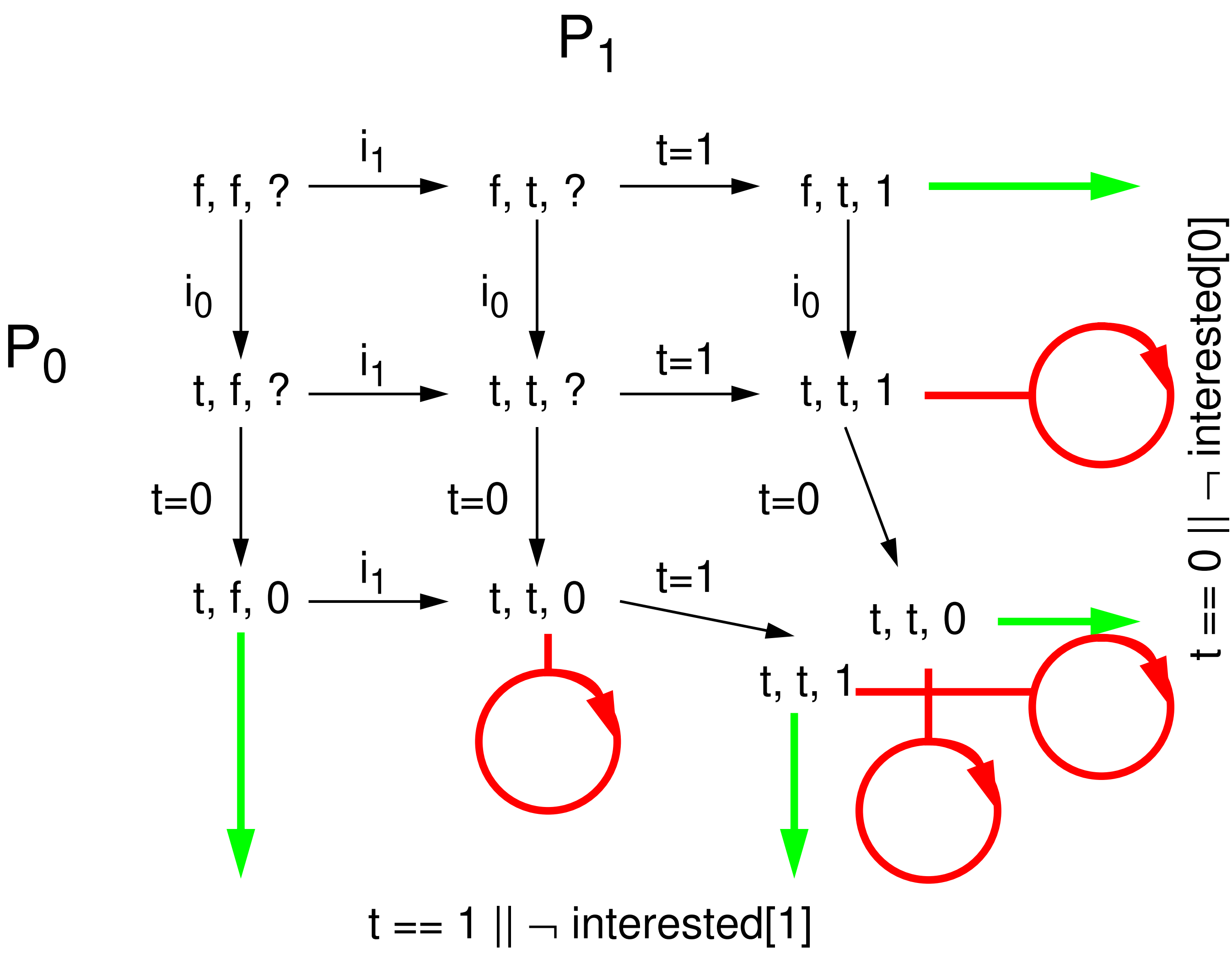
```
thread_0()
{
  while (TRUE) {
    interested[0] = TRUE;
    turn = 0;
    while (turn == 0 && interested[1] == TRUE) ;
    cout << "P 0 ist im krit. Absch.; "
         << "interested = " << interested[0] << ", "
         << interested[1] << "\n" << flush;
    critical_section_0();
    interested[0] = FALSE;
    cout << "P 0 ist ausserhalb krit. Absch.; "
         << "interested = " << interested[0] << ", "
         << interested[1] << "\n" << flush;
    remainder_0();
  }
}
```

- Aktivitätsträger 1

```
thread_1()
{
  while (TRUE) {
    interested[1] = TRUE;
    turn = 1;
    while (turn == 1 && interested[0] == TRUE) ;
    cout << "P 1 ist im krit. Absch.; "
         << "interested = " << interested[0] << ", "
         << interested[1] << "\n" << flush;
    interested[1] = FALSE;
    critical_section_1();
    cout << "P 1 ist ausserhalb krit. Absch.; "
         << "interested = " << interested[0] << ", "
         << interested[1] << "\n" << flush;
    remainder_1();
  }
}
```

```
P 1 ist ausserhalb krit. Absch.; interested = 1, 0
P 0 ist ausserhalb krit. Absch.; interested = 0, 1
P 1 ist im krit. Absch.; interested = 0, 1
P 1 ist ausserhalb krit. Absch.; interested = 1, 0
P 0 ist im krit. Absch.; interested = 1, 1
P 1 ist im krit. Absch.; interested = 0, 1
P 0 ist ausserhalb krit. Absch.; interested = 0, 1
P 1 ist ausserhalb krit. Absch.; interested = 1, 0
P 0 ist im krit. Absch.; interested = 1, 0
P 0 ist ausserhalb krit. Absch.; interested = 0, 1
P 1 ist im krit. Absch.; interested = 1, 1
P 0 ist im krit. Absch.; interested = 1, 0
P 1 ist ausserhalb krit. Absch.; interested = 1, 0
P 0 ist ausserhalb krit. Absch.; interested = 0, 1
P 1 ist im krit. Absch.; interested = 1, 1
P 0 ist im krit. Absch.; interested = 1, 0
P 1 ist ausserhalb krit. Absch.; interested = 1, 0
P 0 ist ausserhalb krit. Absch.; interested = 0, 1
P 1 ist im krit. Absch.; interested = 0, 1
P 1 ist ausserhalb krit. Absch.; interested = 1, 0
P 0 ist im krit. Absch.; interested = 1, 0
P 0 ist ausserhalb krit. Absch.; interested = 0, 1
P 1 ist im krit. Absch.; interested = 1, 1
P 0 ist im krit. Absch.; interested = 1, 0
P 1 ist ausserhalb krit. Absch.; interested = 1, 0
```

Nachweis des gegenseitigen Ausschlusses



Lösung durch Hardwareunterstützung: Swap-Befehl

- Wirkung des swap(exchange)-Befehls

```
void swap(register &r, shared int &m)
{ int temp;
  begin_atomic();
    temp = m; m = r; r = temp;
  end_atomic();
};
```

- Initialisierung

```
shared_int flag = 0;
```

- Aktivitätsträger 0

```
thread_0()
{
  register r;
  while (TRUE) {
    while (r = 1, swap(r, flag), r == 1) ;
    critical_section_0();
    flag = 0;
    remainder_0();
  }
}
```

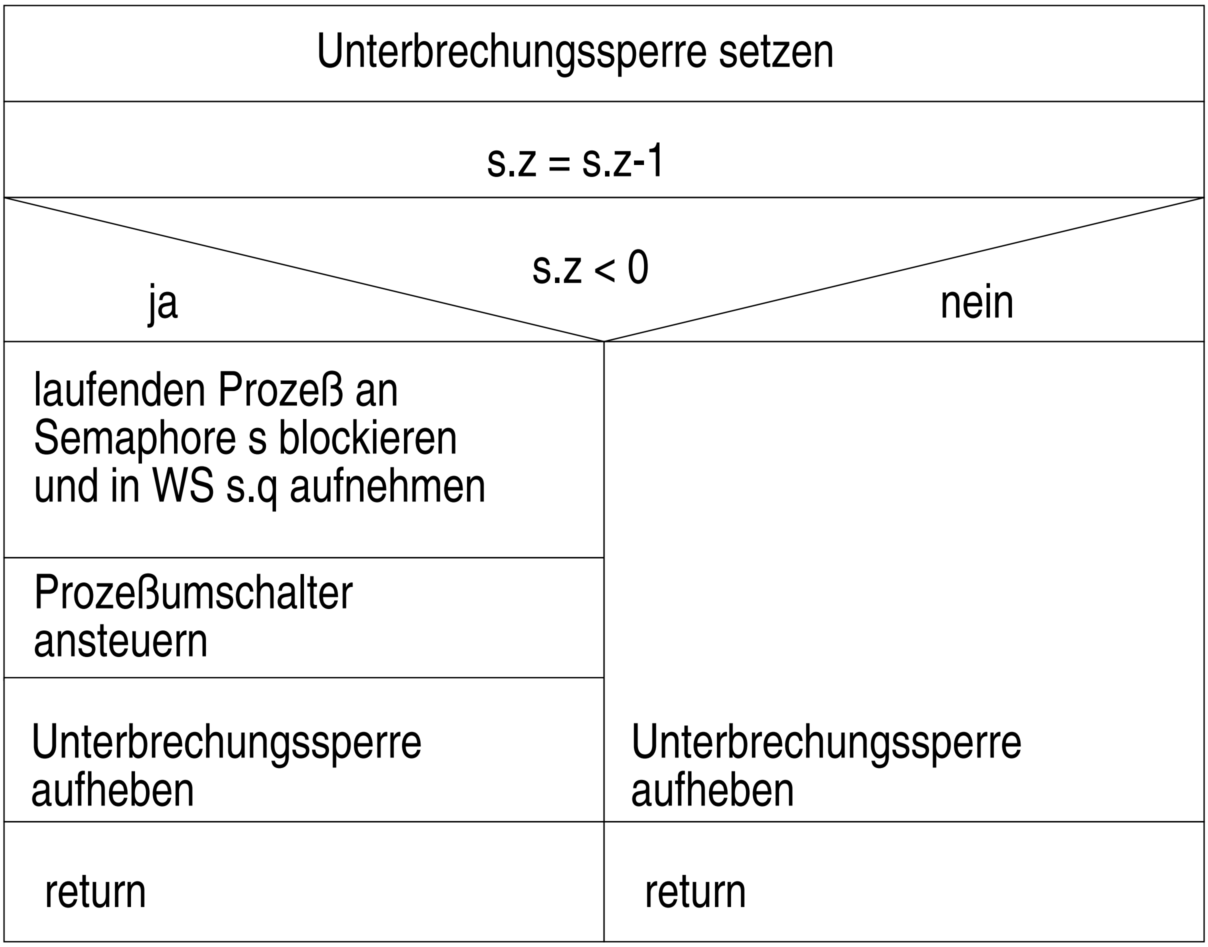
- Aktivitätsträger 1

```
thread_1()
{
  register r;
  while (TRUE) {
    while (r = 1, swap(r, flag), r == 1) ;
    critical_section_1();
    flag = 0;
    remainder_1();
  }
}
```

Lösung für Monoprozessoren: Semaphore

```
typedef  ..........  queue;
class semaphore
  { int z;
    queue q;
  public:
     semaphore() { };
     semaphore(int z0) {z = z0; ... };
     void P();
     void V()
  } s;
```

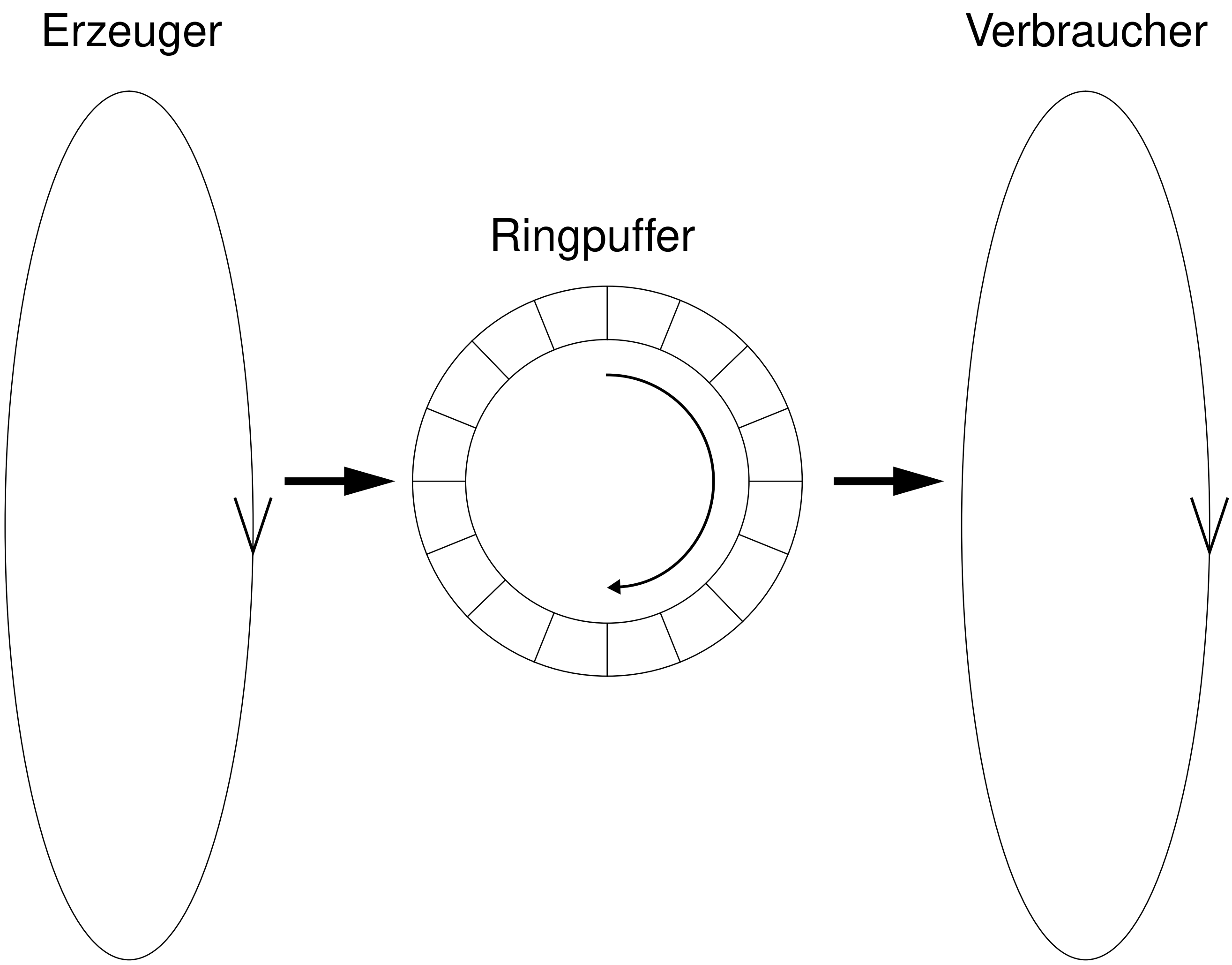
s.P() (Monoprozessor)



s.V() (Monoprozessor)

Unterbrechungssperre setzen	
s.z = s.z+1	
s.z <= 0	
ja	nein
einen Prozeß aus s.q entfernen und in den Zustand bereit versetzen	
Prozeßumschalter ansteuern	
Unterbrechungssperre aufheben	Unterbrechungssperre aufheben
return	return

Erzeuger-Verbraucher mit endlichem Puffer



- Initialisierung

```
semaphore mutex(1), empty(N), full(0);
shared_int buffer[N];
```

- Aktivitätsträger 0 (Erzeuger, Producer)

```
int producer()
{ int value, slot = 0;
  while (TRUE) {
    cout << "Producer: " << flush; cin >> value;
    empty.P();
    mutex.P();
      buffer[slot] = value;
    mutex.V();
    full.V();
    if (++slot == N) slot = 0;
  }
}
```

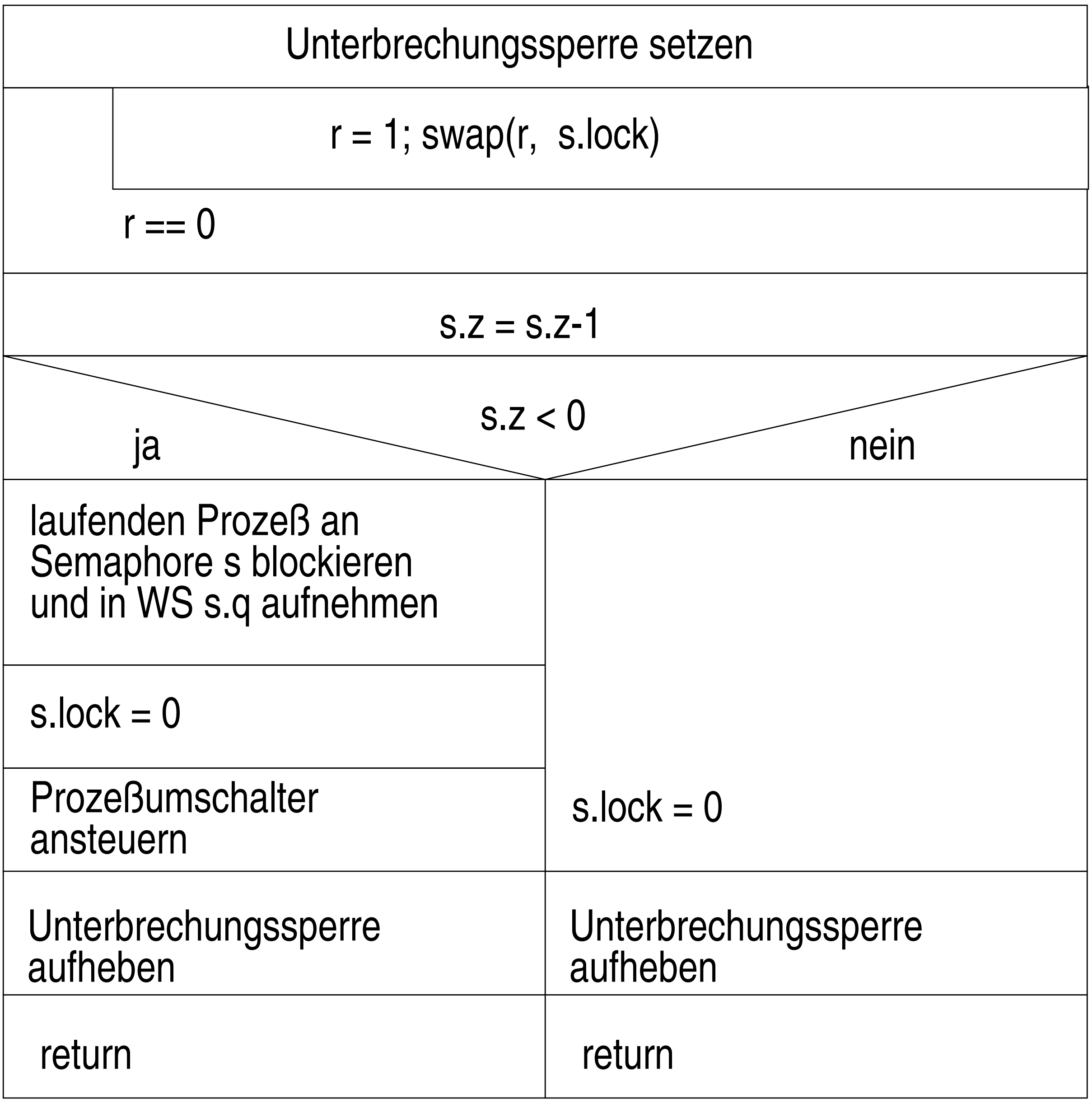
- Aktivitätsträger 1 (Verbraucher, Consumer)

```
int consumer()
{ int value, slot = 0;
  while (TRUE) {
    full.P();
    mutex.P();
      value = buffer[slot];
    mutex.V();
    empty.V();
    if (++slot == N) slot = 0;
    cout << "Consumer: " << value
         << "\n" << flush;
  }
}
```

Semaphore für Multiprozessoren

```
typedef  .........  queue;
class semaphore
  { int z;
    queue q;
    int lock;
  public:
     void P();
     void V()
  } s;
```

s.P()   (Multiprozessor)



s.V() (Multiprozessor)