

Prozess-Synchronisation



Überblick

- Arten der Synchronisation
- Kritische Abschnitte
- Kriterien für Synchronisationslösungen
- Hardware-Unterstützung
- Semaphore
- Klassische Synchronisationsprobleme
- Monitore



Arten der Synchronisation

- Sperrsynchronisation (gegenseitiger Ausschluss, mutual exclusion)
Konkurrenz um die Nutzung gemeinsamer, aber nur exklusiv nutzbarer Betriebsmittel (Variable, Dateien, E/A-Geräte usw.)
- Zustands-/Ereignissynchronisation
Abstimmung der Ausführung der Verarbeitungsschritte aufgrund von Datenabhängigkeiten (z. B. Produzenten-Konsumenten-Synchronisation)

Race Conditions und kritische Abschnitte



Prozess/Thread 1

...

```
i = leseZaehler();  
i = i + 10;  
schreibeZaehler( i );
```

...

Prozess/Thread 2

...

```
j = leseZaehler();  
j = j - 5;  
schreibeZaehler( j );
```

...


Kritische Abschnitte (critical sections):

Programmabschnitte, in denen sich zu jedem Zeitpunkt maximal ein Prozess/Thread befinden darf



Bedingungen für eine gute Lösung (nach Dijkstra)

1. Zu jedem Zeitpunkt befindet sich **höchstens ein** Prozess im kritischen Bereich (**gegenseitiger Ausschluss**)
2. Es werden **keine Annahmen** über die **Ausführungsgeschwindigkeit** oder die **Anzahl** der Prozesse getroffen
3. **Außerhalb** des kritischen Bereichs darf ein Prozess andere Prozesse nicht **blockieren**
4. Keine **unendlich lange** Wartezeit vor dem Eintritt in den kritischen Bereich (**Fairness**)



Erster Lösungsversuch

dran = 1;


Prozess/Thread 1

```
while (dran != 1);  
i = leseZaehler();  
i = i + 10;  
schreibeZaehler( i );  
dran = 2;
```

Prozess/Thread 2

```
while (dran != 2);  
j = leseZaehler();  
j = j - 5;  
schreibeZaehler( j );  
dran = 1;
```

- Probleme:
- Busy-Wait
 - Ein Prozess blockiert **sich selbst**



Zweiter Lösungsversuch

bereit1 = bereit2 = false;

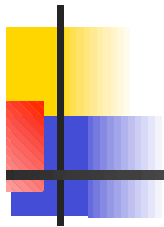
Prozess/Thread 1

```
bereit1 = true;  
while (bereit2 == true);  
i = leseZaehler();  
i = i + 10;  
schreibeZaehler( i );  
bereit1 = false;
```

Prozess/Thread 2

```
bereit2 = true;  
while (bereit1 == true);  
j = leseZaehler();  
j = j - 5;  
schreibeZaehler( j );  
bereit2 = false;
```

- Probleme:
- Busy-Wait
 - Prozesse/Threads können sich gegenseitig blockieren (Verklemmung/Deadlock)



Dritter Lösungsversuch

```
bereit1 = bereit2 = false;  
dran = 0;
```

Prozess/Thread 1

```
bereit1 = true;  
dran = 2;  
while (dran == 2 &&  
    bereit2 == true);  
i = leseZaehler();  
i = i + 10;  
schreibeZaehler( i );  
bereit1 = false;
```

Prozess/Thread 2

```
bereit2 = true;  
dran = 1;  
while (dran == 1 &&  
    bereit1 == true);  
j = leseZaehler();  
j = j - 5;  
schreibeZaehler( j );  
bereit2 = false;
```




Hardware-Unterstützung

- Interrupts ausschalten
- Atomare Maschineninstruktionen:
 - **TestAndSet:**
Auslesen und Setzen einer Speicherzelle
 - **Swap:**
Tauschen des Wertes zweier Speicherzellen
 - **FetchAndAdd:**
Auslesen einer Speicherzelle und Erhöhen der Zelle um einen angegebenen Wert

Synchronisation mit TestAndSet bzw. Swap

belegt = false;

Prozess/Thread

while (TestAndSet(belegt));

i = leseZaehler();

i = i + 10;

schreibeZaehler(i);

belegt = false;

belegt = false;

Prozess/Thread

schluessel = true;

while (schluessel == true)

 Swap(schluessel, belegt);

j = leseZaehler();

j = j - 5;

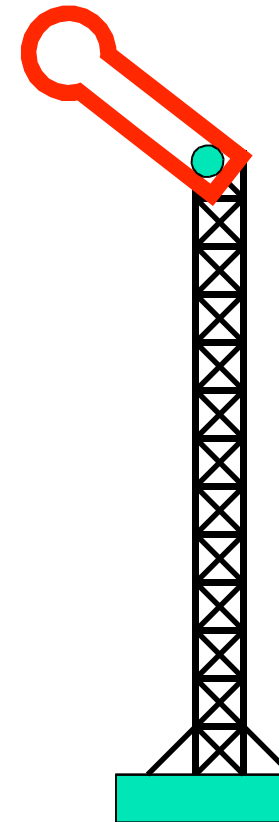
schreibeZaehler(j);

belegt = false;



Semaphore (Dijkstra, 1965)

- „Signalmast“
- Steuervariable S zum Signalisieren des Zustands eines kritischen Abschnitts
- Meist verbunden mit einer zugehörigen Prozess-Warteschlange W
- Atomare (unteilbare, ununterbrechbare) Operationen:
 - S.p(): Passieren der Semaphore (S.wait())
 - S.v(): Verlassen der Semaphore (S.signal())





Lösung mit Semaphor

Prozess/Thread 1

```
S.wait();
```

```
i = leseZaehler();
```

```
i = i + 10;
```

```
schreibeZaehler( i );
```

```
S.signal();
```

Prozess/Thread 2

```
S.wait();
```

```
j = leseZaehler();
```

```
j = j - 5;
```

```
schreibeZaehler( j );
```

```
S.signal();
```

S ist ein Semaphoren-Objekt mit den Methoden
wait() (möchte passieren) und signal() (verlassen)



Realisierung eines binären Semaphors

S.wait():

```
if ( TestAndSet(belegt) ) {  
    Prozess in Warteschlange W einreihen;  
    Prozess in Zustand „wartend“ versetzen;  
}
```

S.signal():

```
if ( W.empty() == false ) {  
    Einen Prozess aus Warteschlange W lösen;  
    Prozess in Zustand „bereit“ versetzen;  
}  
else { belegt = false; }
```



Realisierung eines Zähl-Semaphors

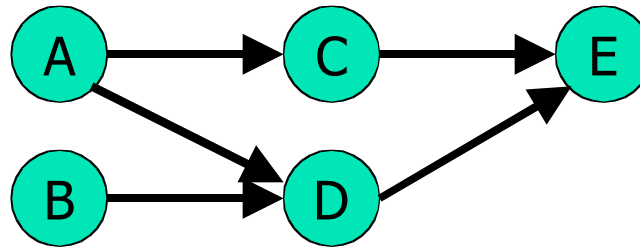
S.wait():

```
if ( FetchAndAdd( zaehler, -1 ) < 1 ) {  
    Prozess in Warteschlange W einreihen;  
    Prozess in Zustand „wartend“ versetzen;  
}
```

S.signal():

```
if ( FetchAndAdd( zaehler, 1 ) < 0 ) {  
    Einen Prozess aus Warteschlange W lösen;  
    Gelösten Prozess in Zustand „bereit“ versetzen;  
}
```

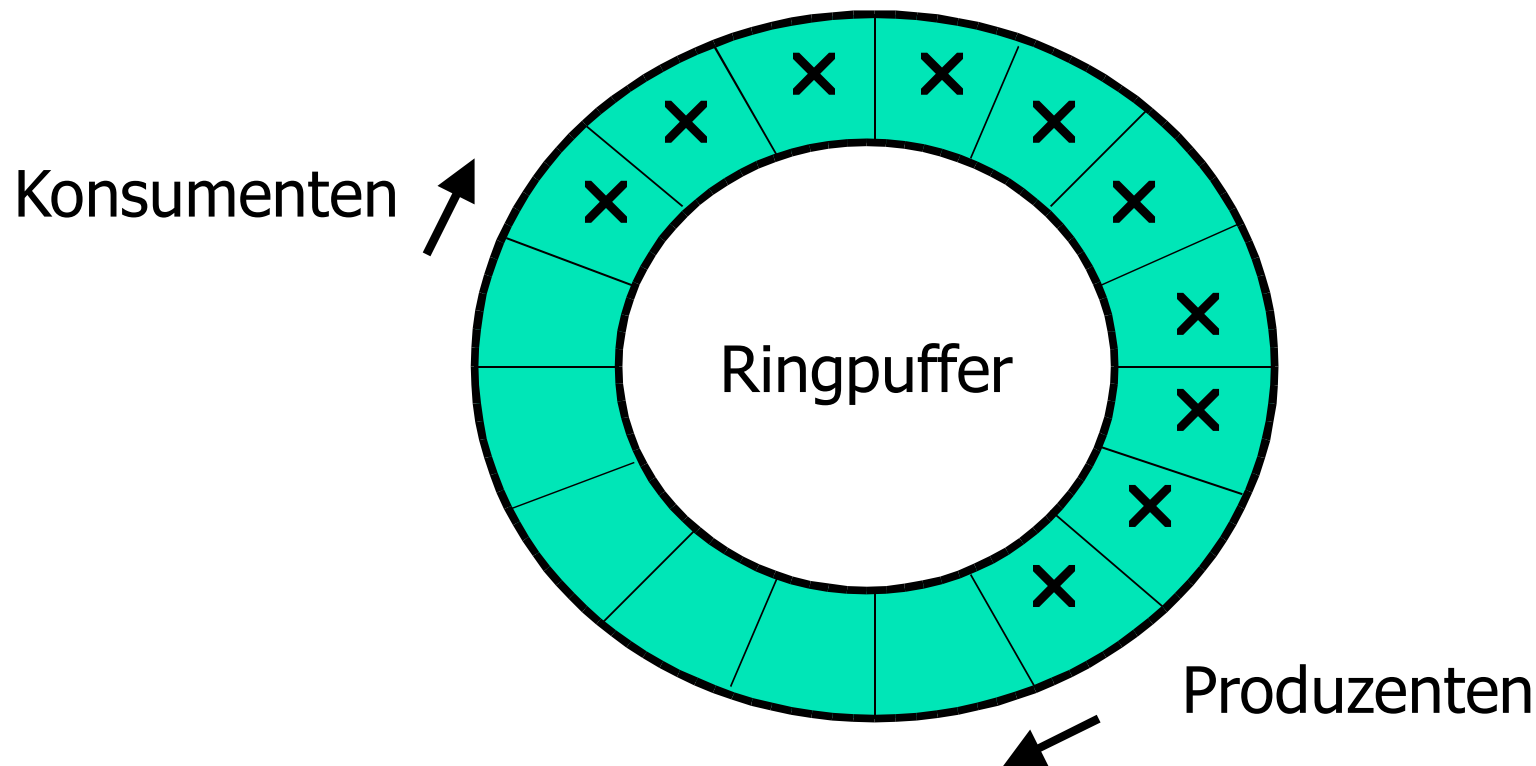
Semaphore zur Synchronisation bei Präzedenzen



Binäre Semaphore c , $d1$, $d2$, $e1$ und $e2$ mit false initialisieren

A:	BodyA;	$c.\text{signal}()$; $d1.\text{signal}()$;
B:	BodyB;	$d2.\text{signal}()$;
C:	$c.\text{wait}()$;	BodyC; $e1.\text{signal}()$;
D:	$d1.\text{wait}()$; $d2.\text{wait}()$;	BodyD; $e2.\text{signal}()$;
E:	$e1.\text{wait}()$; $e2.\text{wait}()$;	BodyE;

Produzenten/Konsumenten-Problem (mit Ringpuffer)



Semaphore zur Produzenten/ Konsumenten-Synchronisation

Produzent

```
while (true) {  
    ware = produziere();  
    frei.wait();  
    mutex.wait();  
    schreibeInPuffer(ware);  
    mutex.signal();  
    belegt.signal();  
}
```

Konsument

```
while (true) {  
    belegt.wait();  
    mutex.wait();  
    ware = holeAusPuffer();  
    mutex.signal();  
    frei.signal();  
    konsumiere(ware);  
}
```

Initialisierung: mutex.zaehler = 1; frei.zaehler = max;
belegt.zaehler = 0;



Weitere klassische Synchronisationsprobleme

- Readers-Writers-Problem
 - Einige Prozesse/Threads wollen einen Datenbereich lesen, einige wollen ihn verändern.
 - Gleichzeitiger Lesezugriff ist erlaubt
 - Schreibzugriffe müssen exklusiv erfolgen
- Dining-Philosophers-Problem
 - Fünf Philosophen sitzen um einen runden Tisch, denken nach und essen Reis mit Stäbchen.
 - Zwischen den Tellern liegt jeweils ein Stäbchen, zum Essen braucht man aber zwei.



Monitore

- **Programmiersprachliches** Konstrukt, funktional äquivalent zu Semaphoren
- Kritische Methoden und Daten werden in einer Klasse mit einem zugehörigen Semaphor kombiniert.
- **Leichter** zu handhaben, weniger fehleranfällig
- Unterstützung der Synchronisation durch **Bedingungsvariablen**

Schematischer Aufbau eines Monitors

