Echtzeitbetriebssysteme

Oliver Jack

Ernst-Abbe-Hochschule Jena
Fachbereich Elektrotechnik und Informationstechnik

Sommersemester 2025

Ernst-Abbe-Hochschule Jena
University of Applied Sciences

Lerneinheit 3. Prozessinteraktion und Deadlock

Lerneinheit 3. Prozessinteraktion und Deadlock

Lernziele

Bitte beachten

Diese Lerneinheit frischt einige wichtige Aspekte des vorangegangenen Teilmoduls *Betriebssysteme* auf.

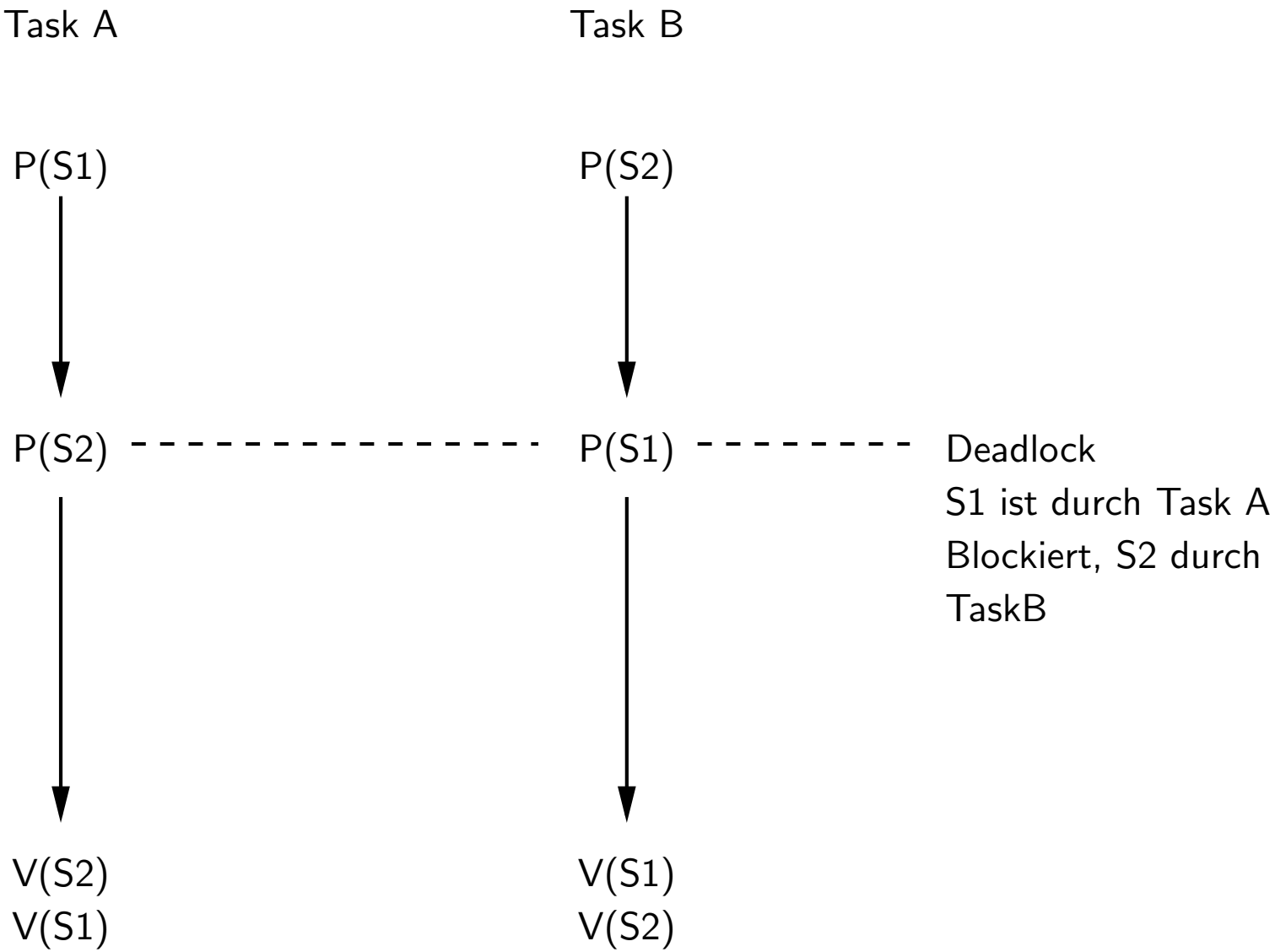
- Verständnis der Deadlock-Eigenschaft
- Kenntnis von Deadlock-Vermeidungsverfahren
- Kenntnis von Prozess-Koordinations- und Kooperationsoperationen

Lerneinheit 3. Prozessinteraktion und Deadlock

Deadlock

- Durch critical sections (gegenseitigem Ausschluss) kann es leicht zu Verklemmungen, den sogenannten Deadlocks kommen.
- Muss eine Task beispielsweise zwei Datenstrukturen manipulieren, die durch zwei unabhängige Semaphore geschützt sind, und eine zweite Task muss das gleiche tun, nur dass sie die Semaphore in umgekehrter Reihenfolge alloziert, gibt es eine Verklemmung (Deadlock).

Deadlock (Forts.)



Vermeidung von Deadlocks

Deadlocks lassen sich durch zwei Maßnahmen vermeiden:

- Entweder durch geeignete Systemauslegung und Programmierung oder dadurch, dass
- nur dann Anforderungen an Betriebsmittel befriedigt werden, wenn sichergestellt ist, dass durch die Anforderung keine Deadlock-Situation entstehen kann.

Zur Überprüfung, ob es zu einer Deadlock-Situation kommen kann, müssen bekannt sein:

- die im System vorhandenen Ressourcen (Betriebsmittelvektor R_{ges}),
- die im System davon bereits auf die einzelnen Tasks verteilten Ressourcen (Belegungsmatrix U) und
- pro Task die maximalen Anforderungen (M).

Deadlock-Sicherheit

- Man bezeichnet den Zustand des Systems solange als deadlock-sicher, solange mindestens eine Task alle Ressourcenanforderungen erfüllen kann.
- Das System gerät dagegen in einen unsicheren Zustand bezüglich Verklemmungen, wenn durch die Anforderung der Task nicht wenigstens eine Task existiert, deren Anforderungen nicht mehr erfüllt werden kann.

Algorithmus zur Überprüfung der Deadlock-Sicherheit

Überprüfung, ob ein System zu einem Zeitpunkt deadlock-sicher ist

Eingabedaten

- Betriebsmittel-Vektor (gibt an, wieviele Betriebsmittel vorhanden sind)
- Gegenwärtige BM-Belegung (also welche Task belegt welche Betriebsmittel in welcher Anzahl)
- Maximale Anforderung an die Betriebsmittel für jede Task

1. Erstelle eine sogenannte Anforderungsmatrix durch Subtraktion der belegten Betriebsmittel von den maximalen Anforderungen für jede Task. Die Anforderungsmatrix gibt also an, wie viele Betriebsmittel eine jede Task zum Zeitpunkt der Untersuchung maximal anfordert.

Algorithmus zur Überprüfung der Deadlock-Sicherheit (Forts.)

2. Es werden die noch zur Verfügung stehenden Betriebsmittel berechnet (BM-Rest-Vektor):
   - Bestimme die Gesamtzahl der belegten Betriebsmittel (R_{used}) durch Summieren der belegten Betriebsmittel.
   - Subtrahiere die Gesamtzahl der belegten Betriebsmittel von den insgesamt vorhandenen Betriebsmitteln.
3. Wenn es in der Anforderungsmatrix M keine Task T_i gibt, bei der alle Anforderungen erfüllt werden können, ist das System in einem unsicheren Zustand bezüglich deadlocks und die aktuelle Anforderung eines Prozesses darf nicht erfüllt werden.

Algorithmus zur Überprüfung der Deadlock-Sicherheit (Forts.)

4. Wenn es in der Anforderungsmatrix M eine Task T_i gibt, bei der alle Anforderungen erfüllt werden können, wird angenommen, dass diese Task zu Ende läuft und alle seine Betriebsmittel wieder freigibt. Folglich können die zur Task gehörenden Betriebsmittel aus der Tabelle der belegten Betriebsmittel gelöscht und den freien Betriebsmitteln zugeordnet werden.
5. Die Schritte 2 und 3 werden solange mit den jeweils aktuellen Werten wiederholt, bis entweder alle Prozesse abgearbeitet werden konnten (sicherer Zustand), oder kein Prozess existiert, dessen maximalen Anforderungen befriedigt werden könnten.

Beispiel Überprüfung der Deadlock-Sicherheit

- System mit den drei Betriebsmittelklassen A, B und C.
- Die Betriebsmittel innerhalb der Klassen sind gemäß folgendem Betriebsmittelvektor R_{ges} mehrfach vorhanden:

A	B	C
10	5	7

- gegenwärtige Verwendung der Betriebsmittel U

Task/BM	A	B	C
T0	0	1	0
T1	2	0	0
T2	3	0	2
T3	2	1	1
T4	0	0	2

Beispiel Überprüfung der Deadlock-Sicherheit (Forts.)

Die einzelnen Tasks stellen maximal die folgenden Anforderungen M

Task/BM	A	B	C
T0	7	5	3
T1	3	2	2
T2	9	0	2
T3	2	2	2
T4	4	3	3

Beispiel Überprüfung der Deadlock-Sicherheit (Forts.)

1. Bestimmung der freien Betriebsmittel. Dazu werden zunächst die belegten Betriebsmittel addiert und von den insgesamt zur Verfügung stehenden Betriebsmittel subtrahiert:

Task/BM	A	B	C
T0	0	1	0
T1	2	0	0
T2	3	0	2
T3	2	1	1
T4	0	0	2
$\sum$	7	2	5

Die freien Betriebsmittel (Betriebsmittel-Rest-Vektor U_{free}) ergeben sich zu:

Beispiel Überprüfung der Deadlock-Sicherheit (Forts.)

1. Bestimmung der freien Betriebsmittel. Dazu werden zunächst die belegten Betriebsmittel addiert und von den insgesamt zur Verfügung stehenden Betriebsmittel subtrahiert:

Task/BM	A	B	C
T0	0	1	0
T1	2	0	0
T2	3	0	2
T3	2	1	1
T4	0	0	2
$\sum$	7	2	5

Die freien Betriebsmittel (Betriebsmittel-Rest-Vektor U_{free}) ergeben sich zu:

A	B	C
10	5	7

–

Beispiel Überprüfung der Deadlock-Sicherheit (Forts.)

1. Bestimmung der freien Betriebsmittel. Dazu werden zunächst die belegten Betriebsmittel addiert und von den insgesamt zur Verfügung stehenden Betriebsmittel subtrahiert:

Task/BM	A	B	C
T0	0	1	0
T1	2	0	0
T2	3	0	2
T3	2	1	1
T4	0	0	2
$\sum$	7	2	5

Die freien Betriebsmittel (Betriebsmittel-Rest-Vektor U_{free}) ergeben sich zu:

A	B	C
10	5	7

$-$

A	B	C
7	2	5

Beispiel Überprüfung der Deadlock-Sicherheit (Forts.)

1. Bestimmung der freien Betriebsmittel. Dazu werden zunächst die belegten Betriebsmittel addiert und von den insgesamt zur Verfügung stehenden Betriebsmittel subtrahiert:

Task/BM	A	B	C
T0	0	1	0
T1	2	0	0
T2	3	0	2
T3	2	1	1
T4	0	0	2
$\sum$	7	2	5

Die freien Betriebsmittel (Betriebsmittel-Rest-Vektor U_{free}) ergeben sich zu:

A	B	C	=	A	B	C	−	A	B	C
3	3	2		10	5	7		7	2	5

Beispiel Überprüfung der Deadlock-Sicherheit (Forts.)

2 Es wird die Anforderungsmatrix M aufgestellt. Dazu wird für jede Task von den maximalen Anforderungen die bereits getätigten Anforderungen subtrahiert:

	A	B	C
T0	7	5	3
T1	3	2	2
T2	9	0	2
T3	2	2	2
T4	4	3	3

Beispiel Überprüfung der Deadlock-Sicherheit (Forts.)

2 Es wird die Anforderungsmatrix M aufgestellt. Dazu wird für jede Task von den maximalen Anforderungen die bereits getätigten Anforderungen subtrahiert:

	A	B	C
T0	7	5	3
T1	3	2	2
T2	9	0	2
T3	2	2	2
T4	4	3	3

$-$

	A	B	C
T0	0	1	0
T1	2	0	0
T2	3	0	2
T3	2	1	1
T4	0	0	2

Beispiel Überprüfung der Deadlock-Sicherheit (Forts.)

2 Es wird die Anforderungsmatrix M aufgestellt. Dazu wird für jede Task von den maximalen Anforderungen die bereits getätigten Anforderungen subtrahiert:

Task/BM	A	B	C
T0	7	4	3
T1	1	2	2
T2	6	0	0
T3	0	1	1
T4	4	3	1

=

	A	B	C
T0	7	5	3
T1	3	2	2
T2	9	0	2
T3	2	2	2
T4	4	3	3

−

	A	B	C
T0	0	1	0
T1	2	0	0
T2	3	0	2
T3	2	1	1
T4	0	0	2

Beispiel Überprüfung der Deadlock-Sicherheit (Forts.)

3 In der Anforderungsmatrix M finden sich zwei Tasks, die mit diesen Betriebsmitteln zu Ende laufen könnten: T1 und T3. Von diesen wird (willkürlich) T1 ausgewählt. Die von T1 belegten Ressourcen (2, 0, 0) werden auf den Vektor U_{free} addiert. Aus der Belegungsmatrix wird die Task T1 gelöscht.

M

Task/BM	A	B	C
T0	7	4	3
T1	1	2	2
T2	6	0	0
T3	0	1	1
T4	4	3	1

U

Task/BM	A	B	C
T0	0	1	0
T1	2	0	0
T2	3	0	2
T3	2	1	1
T4	0	0	2

U_{free}

A	B	C
3	3	2

A	B	C

Beispiel Überprüfung der Deadlock-Sicherheit (Forts.)

3 In der Anforderungsmatrix M finden sich zwei Tasks, die mit diesen Betriebsmitteln zu Ende laufen könnten: T1 und T3. Von diesen wird (willkürlich) T1 ausgewählt. Die von T1 belegten Ressourcen (2, 0, 0) werden auf den Vektor U_{free} addiert. Aus der Belegungsmatrix wird die Task T1 gelöscht.

M

Task/BM	A	B	C
T0	7	4	3
T1	1	2	2
T2	6	0	0
T3	0	1	1
T4	4	3	1

U

Task/BM	A	B	C
T0	0	1	0
T1	2	0	0
T2	3	0	2
T3	2	1	1
T4	0	0	2

U_{free}

A	B	C
3	3	2

A	B	C

Beispiel Überprüfung der Deadlock-Sicherheit (Forts.)

3 In der Anforderungsmatrix M finden sich zwei Tasks, die mit diesen Betriebsmitteln zu Ende laufen könnten: T1 und T3. Von diesen wird (willkürlich) T1 ausgewählt. Die von T1 belegten Ressourcen (2, 0, 0) werden auf den Vektor U_{free} addiert. Aus der Belegungsmatrix wird die Task T1 gelöscht.

M

Task/BM	A	B	C
T0	7	4	3
T1	1	2	2
T2	6	0	0
T3	0	1	1
T4	4	3	1

U

Task/BM	A	B	C
T0	0	1	0
T1	2	0	0
T2	3	0	2
T3	2	1	1
T4	0	0	2

U_{free}

A	B	C
3	3	2

A	B	C

Beispiel Überprüfung der Deadlock-Sicherheit (Forts.)

3 In der Anforderungsmatrix M finden sich zwei Tasks, die mit diesen Betriebsmitteln zu Ende laufen könnten: T1 und T3. Von diesen wird (willkürlich) T1 ausgewählt. Die von T1 belegten Ressourcen (2, 0, 0) werden auf den Vektor U_{free} addiert. Aus der Belegungsmatrix wird die Task T1 gelöscht.

M

Task/BM	A	B	C
T0	7	4	3
T1	1	2	2
T2	6	0	0
T3	0	1	1
T4	4	3	1

U

Task/BM	A	B	C
T0	0	1	0
T1	2	0	0
T2	3	0	2
T3	2	1	1
T4	0	0	2

U_{free}

A	B	C
3	3	2

↓

A	B	C

Beispiel Überprüfung der Deadlock-Sicherheit (Forts.)

3 In der Anforderungsmatrix M finden sich zwei Tasks, die mit diesen Betriebsmitteln zu Ende laufen könnten: T1 und T3. Von diesen wird (willkürlich) T1 ausgewählt. Die von T1 belegten Ressourcen (2, 0, 0) werden auf den Vektor U_{free} addiert. Aus der Belegungsmatrix wird die Task T1 gelöscht.

M

Task/BM	A	B	C
T0	7	4	3
T1	1	2	2
T2	6	0	0
T3	0	1	1
T4	4	3	1

U

Task/BM	A	B	C
T0	0	1	0
T1	2	0	0
T2	3	0	2
T3	2	1	1
T4	0	0	2

U_{free}

A	B	C
3	3	2

↓

A	B	C
5	3	2

Beispiel Überprüfung der Deadlock-Sicherheit (Forts.)

4 Mit dem neuen Belegungsvektor lassen sich entweder die Task T3 oder T4 zu ende rechnen (siehe Anforderungsmatrix). Wieder wird eine Task ausgewählt, hier T4 (0, 0, 2) und gelöscht. Damit ergibt sich ein neuer Betriebsmittel-Rest-Vektor:

M

Task/BM	A	B	C
T0	7	4	3
T2	6	0	0
T3	0	1	1
T4	4	3	1

U

Task/BM	A	B	C
T0	0	1	0
T2	3	0	2
T3	2	1	1
T4	0	0	2

U_{free}

A	B	C
5	3	2

A	B	C

Beispiel Überprüfung der Deadlock-Sicherheit (Forts.)

4 Mit dem neuen Belegungsvektor lassen sich entweder die Task T3 oder T4 zu ende rechnen (siehe Anforderungsmatrix). Wieder wird eine Task ausgewählt, hier T4 (0, 0, 2) und gelöscht. Damit ergibt sich ein neuer Betriebsmittel-Rest-Vektor:

M

Task/BM	A	B	C
T0	7	4	3
T2	6	0	0
T3	0	1	1
T4	4	3	1

U

Task/BM	A	B	C
T0	0	1	0
T2	3	0	2
T3	2	1	1
T4	0	0	2

U_{free}

A	B	C
5	3	2

A	B	C

Beispiel Überprüfung der Deadlock-Sicherheit (Forts.)

4 Mit dem neuen Belegungsvektor lassen sich entweder die Task T3 oder T4 zu ende rechnen (siehe Anforderungsmatrix). Wieder wird eine Task ausgewählt, hier T4 (0, 0, 2) und gelöscht. Damit ergibt sich ein neuer Betriebsmittel-Rest-Vektor:

M

Task/BM	A	B	C
T0	7	4	3
T2	6	0	0
T3	0	1	1
T4	4	3	1

U

Task/BM	A	B	C
T0	0	1	0
T2	3	0	2
T3	2	1	1
T4	0	0	2

U_{free}

A	B	C
5	3	2

↓

A	B	C

Beispiel Überprüfung der Deadlock-Sicherheit (Forts.)

4 Mit dem neuen Belegungsvektor lassen sich entweder die Task T3 oder T4 zu ende rechnen (siehe Anforderungsmatrix). Wieder wird eine Task ausgewählt, hier T4 (0, 0, 2) und gelöscht. Damit ergibt sich ein neuer Betriebsmittel-Rest-Vektor:

M

Task/BM	A	B	C
T0	7	4	3
T2	6	0	0
T3	0	1	1
T4	4	3	1

U

Task/BM	A	B	C
T0	0	1	0
T2	3	0	2
T3	2	1	1
T4	0	0	2

U_{free}

A	B	C
5	3	2

↓

A	B	C
5	3	4

Beispiel Überprüfung der Deadlock-Sicherheit (Forts.)

5 Im nächsten Schritt wird Task T3 (2, 1, 1) ausgewählt (von den in der Tabelle noch existierenden Tasks T0, T2 und T3) und gelöscht. Der neue Betriebsmittel-Rest-Vektor ergibt sich damit zu:

M

Task/BM	A	B	C
T0	7	4	3
T2	6	0	0
T3	0	1	1

U

Task/BM	A	B	C
T0	0	1	0
T2	3	0	2
T3	2	1	1

U_{free}

A	B	C
5	3	4

A	B	C

Beispiel Überprüfung der Deadlock-Sicherheit (Forts.)

5 Im nächsten Schritt wird Task T3 (2, 1, 1) ausgewählt (von den in der Tabelle noch existierenden Tasks T0, T2 und T3) und gelöscht. Der neue Betriebsmittel-Rest-Vektor ergibt sich damit zu:

M

Task/BM	A	B	C
T0	7	4	3
T2	6	0	0
T3	0	1	1

U

Task/BM	A	B	C
T0	0	1	0
T2	3	0	2
T3	2	1	1

U_{free}

A	B	C
5	3	4

A	B	C

Beispiel Überprüfung der Deadlock-Sicherheit (Forts.)

5 Im nächsten Schritt wird Task T3 (2, 1, 1) ausgewählt (von den in der Tabelle noch existierenden Tasks T0, T2 und T3) und gelöscht. Der neue Betriebsmittel-Rest-Vektor ergibt sich damit zu:

M

Task/BM	A	B	C
T0	7	4	3
T2	6	0	0
T3	0	1	1

U

Task/BM	A	B	C
T0	0	1	0
T2	3	0	2
T3	2	1	1

U_{free}

A	B	C
5	3	4

↓

A	B	C
7	4	5

Beispiel Überprüfung der Deadlock-Sicherheit (Forts.)

6 In der Tabelle stehen nur noch die Tasks T0 und T2, die beide zu Ende laufen können. Als nächstes wird T0 ausgewählt und gelöscht. Der neue Betriebsmittel-Rest-Vektor ergibt sich zu:

M

Task/BM	A	B	C
T0	7	4	3
T2	6	0	0

U

Task/BM	A	B	C
T0	0	1	0
T2	3	0	2

U_{free}

A	B	C
7	4	5

A	B	C

Beispiel Überprüfung der Deadlock-Sicherheit (Forts.)

6 In der Tabelle stehen nur noch die Tasks T0 und T2, die beide zu Ende laufen können. Als nächstes wird T0 ausgewählt und gelöscht. Der neue Betriebsmittel-Rest-Vektor ergibt sich zu:

M

Task/BM	A	B	C
T0	7	4	3
T2	6	0	0

U

Task/BM	A	B	C
T0	0	1	0
T2	3	0	2

U_{free}

A	B	C
7	4	5

A	B	C

Beispiel Überprüfung der Deadlock-Sicherheit (Forts.)

6 In der Tabelle stehen nur noch die Tasks T0 und T2, die beide zu Ende laufen können. Als nächstes wird T0 ausgewählt und gelöscht. Der neue Betriebsmittel-Rest-Vektor ergibt sich zu:

M

Task/BM	A	B	C
T0	7	4	3
T2	6	0	0

U

Task/BM	A	B	C
T0	0	1	0
T2	3	0	2

U_{free}

A	B	C
7	4	5

↓

A	B	C
7	5	5

Beispiel Überprüfung der Deadlock-Sicherheit (Forts.)

7 Jetzt ist nur noch T2 verbleibend. Da auch diese Anforderungen (3, 0, 2) erfüllt werden können, ist das System in einem deadlock-sicheren Zustand.

M

Task/BM	A	B	C
T2	6	0	0

U

Task/BM	A	B	C
T2	3	0	2

U_{free}

A	B	C
7	5	5

A	B	C

Beispiel Überprüfung der Deadlock-Sicherheit (Forts.)

7 Jetzt ist nur noch T2 verbleibend. Da auch diese Anforderungen (3, 0, 2) erfüllt werden können, ist das System in einem deadlock-sicheren Zustand.

M

Task/BM	A	B	C
T2	6	0	0

U

Task/BM	A	B	C
T2	3	0	2

U_{free}

A	B	C
7	5	5

A	B	C

Beispiel Überprüfung der Deadlock-Sicherheit (Forts.)

7 Jetzt ist nur noch T2 verbleibend. Da auch diese Anforderungen (3, 0, 2) erfüllt werden können, ist das System in einem deadlock-sicheren Zustand.

M

Task/BM	A	B	C
T2	6	0	0

U

Task/BM	A	B	C
T2	3	0	2

U_{free}

A	B	C
7	5	5

↓

A	B	C
10	5	7

Lerneinheit 3. Prozessinteraktion und Deadlock

POSIX-Threads (IEEE POSIX-Standard)

Mutex dient dem gegenseitigen Ausschluss

pthread_mutex_init() initialisiert ein Mutex-Objekt

pthread_mutex_lock() wird beim Betreten des kritischen Abschnitts aufgerufen und blockiert den Aufrufer, falls belegt.

pthread_mutex_trylock() ist die nichtblockierende Variante: Falls frei, wird belegt; Falls belegt, wird mit entsprechendem Hinweis zurückgekehrt

pthread_mutex_unlock() beim Verlassen des kritischen Abschnitts

POSIX-Threads (IEEE POSIX-Standard) (Forts.)

Cond (Variable) dient der Signalisierung

pthread_cond_wait() blockiert, falls Signal nicht gesetzt

pthread_cond_timedwait() zusätzlich mit Fristablauf (time-out)

pthread_cond_signal() setzt das Signal und deblockiert den "vordersten" Thread (Priorität bzw. FCFS)

pthread_cond_broadcast() setzt Signal und deblockiert alle wartenden Prozesse

POSIX-Semaphore

Semaphor hilft bei gegenseitigem Ausschluss

sem_init(sem_t *sem, int pshared, unsigned int value) Initialisieren der Semaphore sem Unter Linux: `pshared` = 0 (Semaphore kann nicht zwischen Prozessen geteilt werden) `value`: Wert der Semaphore

sem_wait(sem_t * sem) Warten auf Semaphore (P-Operation)

sem_trywait(sem_t * sem) Würde man auf die Semaphore warten müssen, dann Rückgabewert -1 falls Operation blockieren würde

POSIX-Semaphore (Forts.)

Semaphor hilft bei gegenseitigem Ausschluss

sem_getvalue(sem_t * sem, int * sval) Liefert Wert der Semaphoren-Variable Negativer Rückgabe-Wert: Zahl der wartenden Threads

sem_post(sem_t * sem) Erhöhen der Semaphore (Signal- bzw-V-Operation)

sem_destroy(sem_t * sem) Löschen der Semaphore

Readers-Writers Problem in C

```
#include <pthread.h>
#include <semaphore.h>
#include <stdlib.h>

pthread_attr_t attributes;
pthread_t r1, r2, w1, w2;
sem_t mutex, srmutex, wmutex;
int nreaders;
```

Readers-Writers Problem in C (Forts.)

```
void reader(int *id) {
  while (1) {
    sem_wait(&mutex);
    if (nreaders == 0) {
      nreaders++;
      sem_wait(&wmutex);
    } else {
      nreaders++;
    }
    sem_post(&mutex);
    printf("Thread %i is reading\n", *id);
    sem_wait(&mutex);
    nreaders--;
    if (nreaders == 0) {
      sem_post(&wmutex);
    }
    sem_post(&mutex);
  }
}
```

Readers-Writers Problem in C (Forts.)

```
void writer(int *id) {
  while (1) {
    sem_wait(&srmutex);
    sem_wait(&wmutex);
    printf("Thread␣%i␣is␣writing␣\n", *id);
    sem_post(&wmutex);
    sem_post(&srmutex);
  }
}
```

Readers-Writers Problem in C (Forts.)

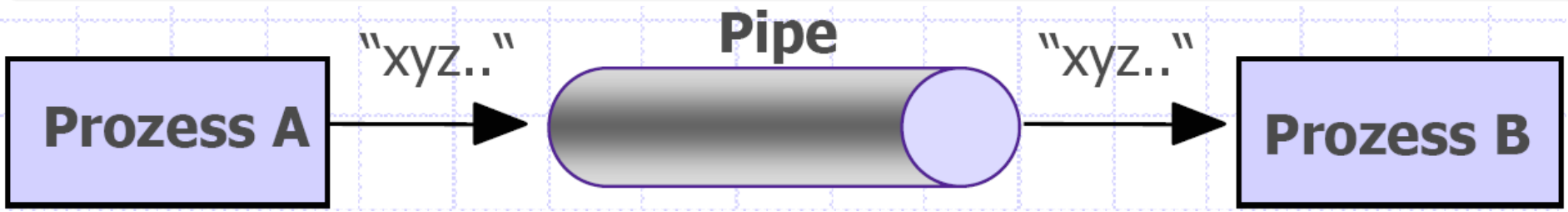
```
int main() {
  int id1 = 1, id2 = 2, id3 = 3, id4 = 5;
  void *result;
  sem_init(&mutex, 0, 1);
  sem_init(&wmutex, 0, 1);
  sem_init(&srmutex, 0, 1);
  if (pthread_attr_init(&attributes) != 0)
    /* set default attributes */
    exit(EXIT_FAILURE);
  if (pthread_create(&r1, &attributes,
      (void *) reader, &id1) != 0)
    exit(EXIT_FAILURE);
```

Readers-Writers Problem in C (Forts.)

```
  if (pthread_create(&r2, &attributes,
        (void *) reader, &id2) != 0)
    exit(EXIT_FAILURE);
  if (pthread_create(&w1, &attributes,
        (void *) writer, &id3) != 0)
    exit(EXIT_FAILURE);
  if (pthread_create(&w2, &attributes,
        (void *) writer, &id4) != 0)
    exit(EXIT_FAILURE);
  pthread_join(r1, (void **) &result);
  /* need to block main program */
  exit(EXIT_FAILURE);
  /* error exit, the program should not terminate */
}
```

Pipe

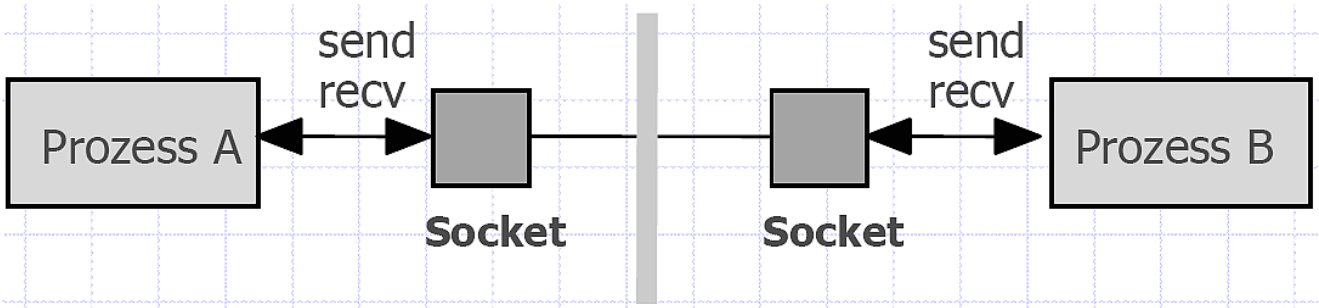
- Spezieller 1:1 Kanal für kontinuierlichen, gerichteten Zeichenstrom
- Die Pipe hat eine begrenzte Kapazität.
- Ist die Pipe voll, so wird ein sendender (schreibender) Prozess blockiert.
- Ist die Pipe leer, so wird ein empfangender (lesender) Prozess blockiert.
- Nur lokaler Mechanismus zwischen genau zwei Prozessen



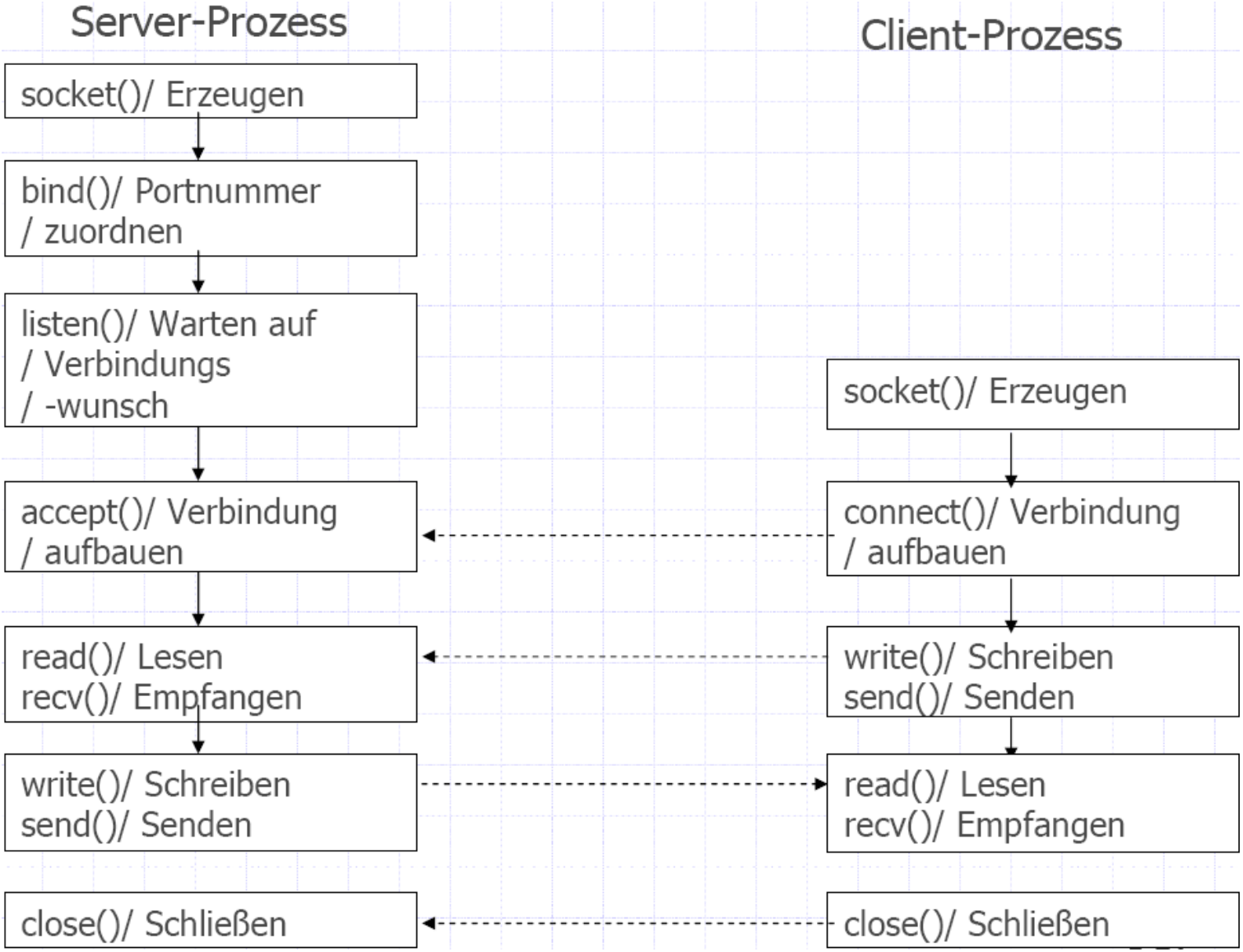
Sockets (Unix, Windows)

- Sockets sind Endpunkte einer Duplex-Verbindung
- Ein Socket kann von mehreren Prozessen benutzt werden
- Verschiedene Socket-Typen werden angeboten

  stream socket verbindungsorientiert
  datagram socket paketorientiert
  raw socket Durchgriff auf zugrundeliegende Protokolle
- Einsatz vor allem zur nichtlokalen Kommunikation (verteilte Systeme)
- Blockierend (synchron) oder nichtblockierend (asynchron)



Socket-Nutzung



Lerneinheit 3. Prozessinteraktion und Deadlock

Zusammenfassung

- Kritische Abschnitte können können zu Systemverklemmungen (Deadlocks) führen.
- Es gibt verschiedene Maßnahmen zur Deadlock-Vermeidung.
- Deadlock-Sicherheit lässt sich (für einen Zeitpunkt) algorithmisch überprüfen.