

• Multithreading


Von Marius Büttner (5346205), Sebastian Buhl (3859293) und
Daniel Seßler (5381517)

INF15A

Am 19.01.2017



● Gliederung

- 
1. Motivation
 2. Probleme
 3. Synchronisation
 4. Modelle



Was ist Multithreading?

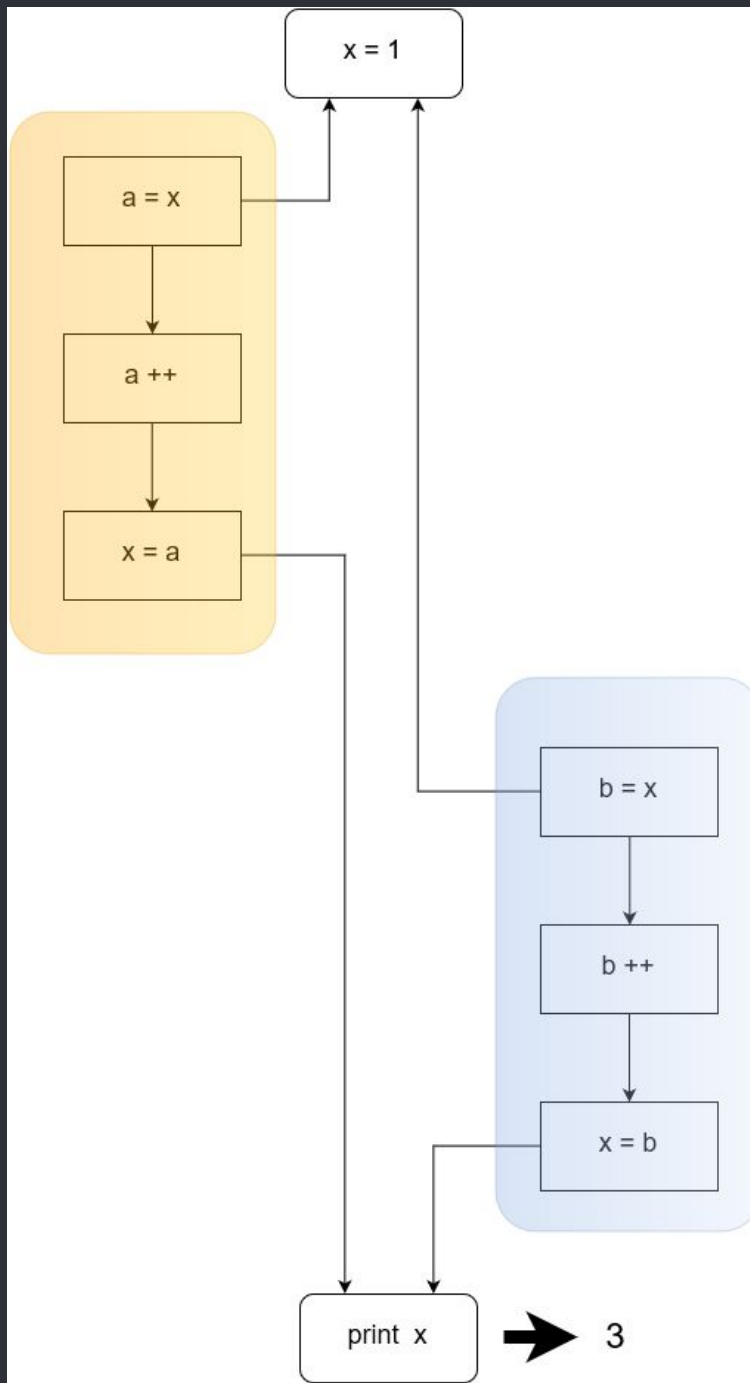


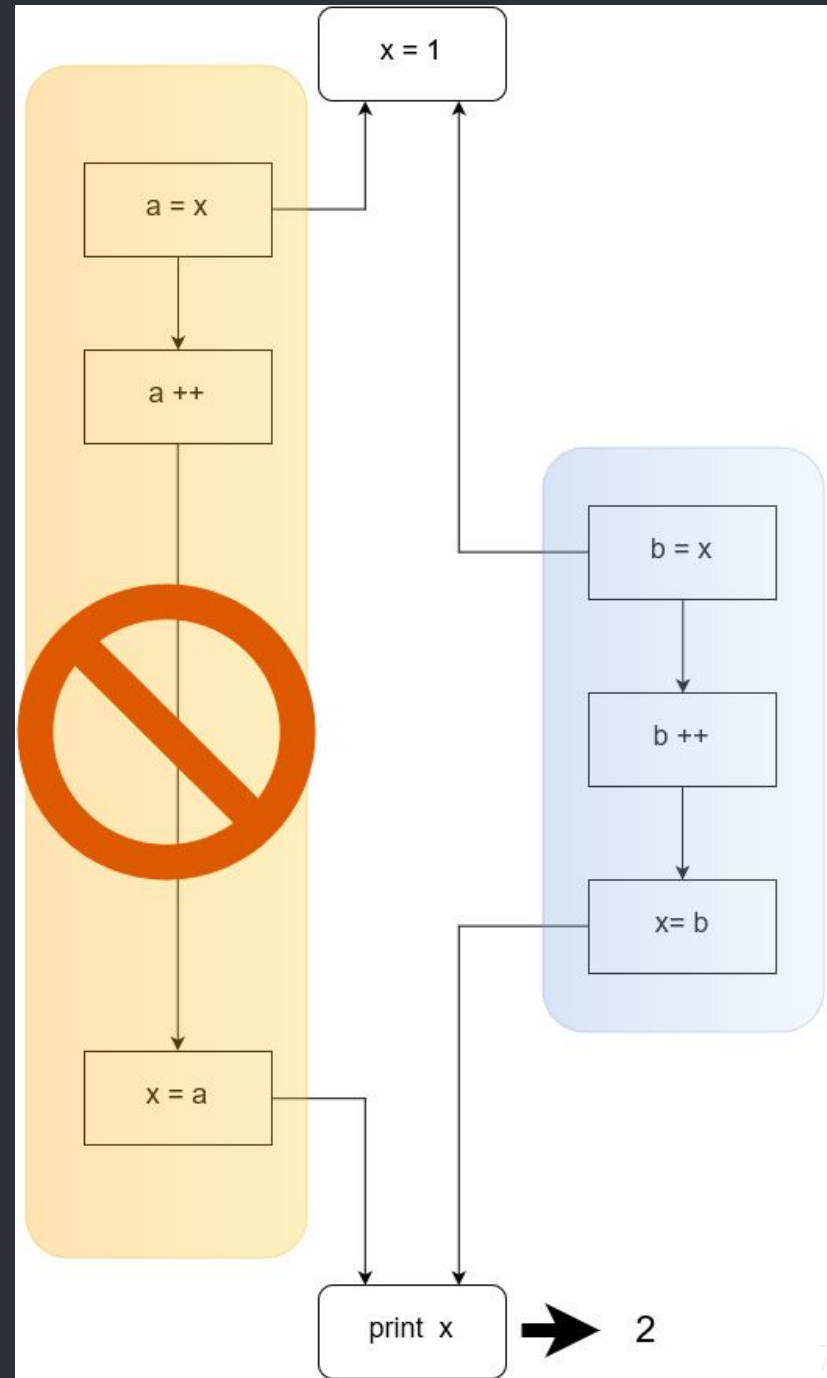
1. Motivation

- Systemausnutzung
- Schnellere Laufzeit
- Bessere Reaktionszeiten
- Parallelisierung

● Race Condition

- “Kritischer Wettlauf”
- Signale konkurrieren um Beeinflussung der Ausgabe

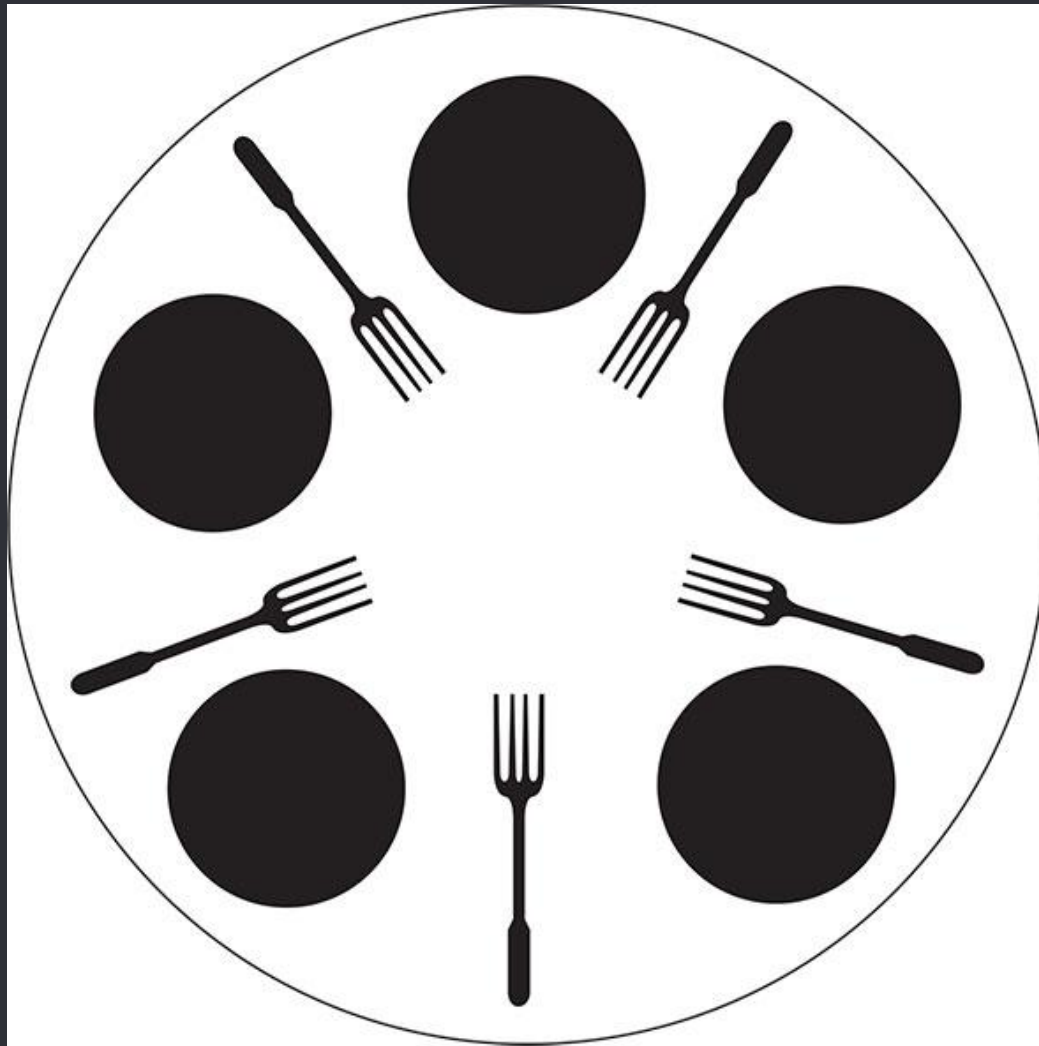




Kritischer Bereich

- Zu einer Zeit nur ein Thread in kritischem Bereich
- Umgang mit kritischen Bereichen
 - Wechselseitiger Ausschluss
 - Fortschritt
 - Begrenzte Wartezeit

● Philosophenproblem



● Deadlock

- Prozesse warten auf ein Ereignis das nur anderer Prozess triggern kann
- Deadlock Kriterien:
 - No Preemption
 - Hold and Wait
 - Mutual Exclusion
 - Circular Wait

● Performance

- single core machines:
 - Threads werden als Queue abgearbeitet
 - Entstehen von Overheads→ keine Optimierung, eher Verlust
- multiple core machines:
 - Hardware unterstützt paralleles Arbeiten
 - Aufteilung der Threads→ Optimierung möglich

3. Synchronisation

- Semaphore
- Mutex
- Monitor
- Lock

3. Synchronisation

- Synchronized
- Volatile
- Lock-Mechanismen
(*ReadWriteLock, ReentrantLock*)
- Atomare Variablen
(*AtomicInteger, AtomicBoolean, ...*)
- Concurrent Collections
(*ConcurrentHashMap, ConcurrentLinkedQueue, ...*)

4. Modelle

- Häufig auftretende Probleme
- Lösungsansätze
- Prozesssynchronisation

● Producer-consumer



- Gemeinsamer Puffer
- Eigenständige Prozesse
- Auf bel. viele Prozesse erweiterbar
- Synchronisation z.B. mithilfe eines Mutex

● Single Write - Multiple Read

- 1 : n Verhältnis -> Ungleichgewicht
- Consumer haben Priorität
- Synchronisation wie zuvor ist ineffizient
- Consumer sollen nicht warten wenn Ressource durch einen Consumer belegt ist

Beispiel Consumer Priority

```
Semaphore mutex;  
Semaphore resource;  
int readcount = 0;  
writer() {  
    resource.lock();  
    //WRITE STUFF  
    resource.unlock();  
}  
reader() {  
    mutex.lock();  
    readcount++;  
    if (readcount == 1)  
        resource.lock(); // Ressource sperren  
    mutex.unlock();  
    //READ STUFF  
    mutex.lock();  
    readcount--;  
    if (readcount == 0)  
        resource.unlock();  
    mutex.unlock();  
}
```

- Multiple Write - Single Read

- - $n : 1$ Verhältnis
 - Producer haben Priorität
 - Umkehrung des ersten Problems

- Multiple Write - Multiple Read

- - Consumer/Producer im Gleichgewicht
 - Kein Priorisierung der Parteien



Quellen

- [https://en.wikipedia.org/w/index.php?title=Synchronization_\(computer_science\)&oldid=759137180](https://en.wikipedia.org/w/index.php?title=Synchronization_(computer_science)&oldid=759137180)
- [https://en.wikipedia.org/w/index.php?title=Thread_\(computing\)&oldid=759459001](https://en.wikipedia.org/w/index.php?title=Thread_(computing)&oldid=759459001)
- https://www.cs.auckland.ac.nz/courses/compsci230s1c/lectures/nasser/21_Multi-threading.pdf
- <http://courses.cs.washington.edu/courses/cse471/11sp/lectures/mtStudent.pdf>
- https://de.wikipedia.org/w/index.php?title=Race_Condition&oldid=160157585
- https://de.wikipedia.org/w/index.php?title=Kritischer_Abschnitt&oldid=156476691
- [https://de.wikipedia.org/w/index.php?title=Deadlock_\(Informatik\)&oldid=161302507](https://de.wikipedia.org/w/index.php?title=Deadlock_(Informatik)&oldid=161302507)