Grundlagen der Rechnerarchitektur und Betriebssysteme

Nebenläufigkeit

Florian Knoch · Lehrstuhl für Praktische Informatik,
insbes. Systemnahe Programmierung, Universität Bamberg

HOCHSCHUL-WAHLEN 2026

Wählt eure Studierendenvertretung!
16. bis 23. Juni 2026

STUVE-BAMBERG.DE/WAHLEN

STUVE BAMBERG

INSTAGRAM: @STUVE_BAMBERG

Input/Output

REVISITED

Verteilte Zahlen einsammeln

Entwurf in Pseudocode

AUFGABE 1A

Betrachten Sie als gegeben:

> Dateien `1.txt`, `2.txt` … `10000.txt` mit je exakt einer positiven Ganzzahl

Gesucht ist die Summe der Zahlen in allen Textdateien. Formulieren Sie eine passende Lösung in Pseudocode.

Verteilte Zahlen einsammeln

Entwurf in Pseudocode

AUFGABE 1A

Betrachten Sie als gegeben:

> Dateien `1.txt`, `2.txt` … `10000.txt` mit je exakt einer positiven Ganzzahl

Gesucht ist die Summe der Zahlen in allen Textdateien. Formulieren Sie eine passende Lösung in Pseudocode.

```
sum = 0;
for (file_number in [1 .. 10000]) {
  file = file_number + ".txt"
  sum = sum + read_number(file);
}
print(sum);
```

Verteilte Zahlen einsammeln

Entwurf in Pseudocode

AUFGABE 1A

Betrachten Sie als gegeben:

> Dateien `1.txt`, `2.txt` ... `10000.txt` mit je exakt einer positiven Ganzzahl

Gesucht ist die Summe der Zahlen in allen Textdateien. Formulieren Sie eine passende Lösung in Pseudocode.

```
sum = 0;
for (file_number in [1 .. 10000]) {
  file = file_number + ".txt"
  sum = sum + read_number(file);
}
print(sum);
```

AUFGABE 2

Implementieren Sie den Algorithmus als C-Programm.

Verteilte Zahlen einsammeln

mit mehreren Threads

AUFGABE 3

1. Handelt es sich hierbei um eine CPU-lastige oder eine I/O-lastige Anwendung?
2.
3.
4.

Verteilte Zahlen einsammeln

mit mehreren Threads

AUFGABE 3

1. Handelt es sich hierbei um eine CPU-lastige oder eine I/O-lastige Anwendung?
2. Wie beeinflusst dies die Zahl der Threads, die zum Einsatz kommen sollten?
3.
4.

Verteilte Zahlen einsammeln

mit mehreren Threads

AUFGABE 3

1. Handelt es sich hierbei um eine CPU-lastige oder eine I/O-lastige Anwendung?
2. Wie beeinflusst dies die Zahl der Threads, die zum Einsatz kommen sollten?
3. Welche realen Anwendungen können sich hinter diesem *Toy Example* verbergen?
4.

Verteilte Zahlen einsammeln

mit mehreren Threads

AUFGABE 3

1. Handelt es sich hierbei um eine CPU-lastige oder eine I/O-lastige Anwendung?
2. Wie beeinflusst dies die Zahl der Threads, die zum Einsatz kommen sollten?
3. Welche realen Anwendungen können sich hinter diesem *Toy Example* verbergen?
4. Wie können wir unseren Code sinnvoll auf mehrere Threads aufteilen?

Threads

LIVE CODING

Verteilte Zahlen einsammeln

Nachgehakt

AUFGABE 4

1. Wie unterscheiden sich die Laufzeiten unserer Programme?
2.
3.
4.

Verteilte Zahlen einsammeln

Nachgehakt

AUFGABE 4

1. Wie unterscheiden sich die Laufzeiten unserer Programme?
2. Wie wirkt sich die Anzahl der Threads auf die Laufzeit aus?
3.
4.

Verteilte Zahlen einsammeln

Nachgehakt

AUFGABE 4

1. Wie unterscheiden sich die Laufzeiten unserer Programme?
2. Wie wirkt sich die Anzahl der Threads auf die Laufzeit aus?
3. Welche anderen Faktoren spielen bei unseren „Messungen“ eine Rolle?
4.

Verteilte Zahlen einsammeln

Nachgehakt

AUFGABE 4

1. Wie unterscheiden sich die Laufzeiten unserer Programme?
2. Wie wirkt sich die Anzahl der Threads auf die Laufzeit aus?
3. Welche anderen Faktoren spielen bei unseren „Messungen“ eine Rolle?
4. Wie können diese Faktoren in wissenschaftlichen Arbeiten eliminiert werden, um Vergleichbarkeit der Ergebnisse zu erzielen?

Verteilte Zahlen einsammeln

N=100.000 Zahlen

	HDD							
	naive	**1 thread**	**2 threads**	**4 threads**	**8 threads**	**16 threads**	**32 threads**	**64 threads**
	5.091	5.052	3.678	2.571	2.639	2.316	2.752	3.632
	4.760	5.399	3.009	2.794	2.660	2.295	2.829	3.602
	4.733	6.164	3.057	2.615	2.626	2.260	2.916	3.679
	5.149	6.159	3.148	2.610	2.640	2.652	3.003	4.280
	6.309	6.123	2.957	2.539	2.620	2.732	3.187	4.383
	6.290	6.176	3.331	2.717	2.629	2.770	3.386	4.378
	6.194	5.023	3.766	2.620	2.628	2.664	2.899	4.277
	6.188	4.867	3.774	2.471	2.744	2.672	2.797	4.360
	5.473	5.147	3.639	2.212	2.694	2.651	2.839	4.353
	5.010	6.098	3.768	2.309	2.453	2.646	2.703	4.362
	5.099	6.291	3.673	2.330	2.246	2.656	2.784	4.431
Mittelwert	**5.678**	**5.739**	**3.502**	**2.528**	**2.550**	**2.567**	**2.815**	**4.176**
Vergleich		**1.07 %**	**-38.32 %**	**-55.47 %**	**-55.10 %**	**-54.80 %**	**-50.42 %**	**-26.45 %**
MIN	4.733	4.795	2.910	2.156	2.221	2.228	2.636	3.504
MAX	6.530	7.358	3.992	2.938	2.798	2.865	3.657	5.228
SD	0.566	0.585	0.319	0.179	0.174	0.180	0.170	0.369

Shared Memory

Arbeit mit geteilten Daten

Hintergrund

- bisher: Ergebnis wird am Ende jedes Threads an den Prozess zurückgegeben
- Threads müssen oftmals auch während der Ausführung kommunizieren
- Möglichkeiten: **Shared Memory**, Message Passing, ...

AUFGABE 5

Nutzen Sie einen zwischen allen Threads geteilten Speicherbereich zum Akkumulieren des Ergebnisses.

Arbeit mit geteilten Daten

Semaphore [1] als Synchronisierungsmechanismus

- **Initialwert**: bestimmt, wie viele Prozesse gleichzeitig die Ressource nutzen können
- **Operation P** (nl. *Probeer te verlagen*): Wert absenken, um Zugriff zu erhalten
  `sem.wait()` oder `acquire()`
- **Operation V** (nl. *Verhoog*): Wert erhöhen, Lock freigeben
  `sem.post()`, `signal()` oder `release()`

AUFGABE 6

Nutzen Sie einen Semaphor, um die Speicherzugriffe zu schützen.

Arbeit mit geteilten Daten

Semaphore [1] als Synchronisierungsmechanismus

- **Initialwert**: bestimmt, wie viele Prozesse gleichzeitig die Ressource nutzen können
- **Operation P** (nl. *Probeer te verlagen*): Wert absenken, um Zugriff zu erhalten
  `sem.wait()` oder `acquire()`
- **Operation V** (nl. *Verhoog*): Wert erhöhen, Lock freigeben
  `sem.post()`, `signal()` oder `release()`

AUFGABE 6

Nutzen Sie einen Semaphor, um die Speicherzugriffe zu schützen.

AUFGABE 7

Welche Alternativen zu Semaphoren gibt es, um unsere Zugriffe auf geteilte Speicherzugriffe zu synchronisieren?

Arbeit mit geteilten Daten

Lockfreie Synchronisation

AUSGANGSPUNKT

Hochsprachenkonstrukt für
eine atomare Operation

```
atomic_int n = 42;
void addTwentyThree() {
  atomic_fetch_add(&n, 23);
}
```

Arbeit mit geteilten Daten

Lockfreie Synchronisation

AUSGANGSPUNKT

Hochsprachenkonstrukt für eine atomare Operation

```
atomic_int n = 42;
void addTwentyThree() {
  atomic_fetch_add(&n, 23);
}
```

VARIANTE 1

Compiler fügt bei der Übersetzung ein Lock ein

```
__atomic_lock(&n);
n += 23;
__atomic_unlock(&n);
```

Arbeit mit geteilten Daten

Lockfreie Synchronisation

AUSGANGSPUNKT

Hochsprachenkonstrukt für eine atomare Operation

```
atomic_int n = 42;
void addTwentyThree() {
  atomic_fetch_add(&n, 23);
}
```

VARIANTE 1

Compiler fügt bei der Übersetzung ein Lock ein

```
__atomic_lock(&n);
n += 23;
__atomic_unlock(&n);
```

VARIANTE 2

Compiler verwendet atomare Instruktionen

```
amoadd x1, x2, x3
```

Fragen?

Literatur

[1] E. W. Dijkstra, „Cooperating Sequential Processes“, EWD 123, 1968. [Online]. Verfügbar unter: **https://www.cs.utexas.edu/users/EWD/ewd01xx/EWD123.PDF**

Lizenzhinweise

Kursmaterialien · **Quelltext**

Diese Präsentation wurde zuletzt am 12.06.2026 bearbeitet. Sie basiert auf den Folien der zugehörigen Vorlesung von Prof. Dr. Michael Engel. Sofern nicht anders angegeben, sind die Inhalte unter der **Lizenz CC BY-SA 4.0** verfügbar. Das Universitätslogo sowie die Schriftart UB Scala Sans sind Eigentum der Otto-Friedrich-Universität Bamberg.

Folgende **Open-Source-Komponenten** kommen in der Präsentation zum Einsatz: **circuiteria** (Apache 2.0), **finite** (MIT), **fontawesome** (SIL), **pgf-tikz** (GPL 2.0), **typst** (Apache 2.0), **typst-ccicons** (MIT), **typst-polylux** (MIT).