

Threading

Ingo Köster

Diplom Informatiker (FH)

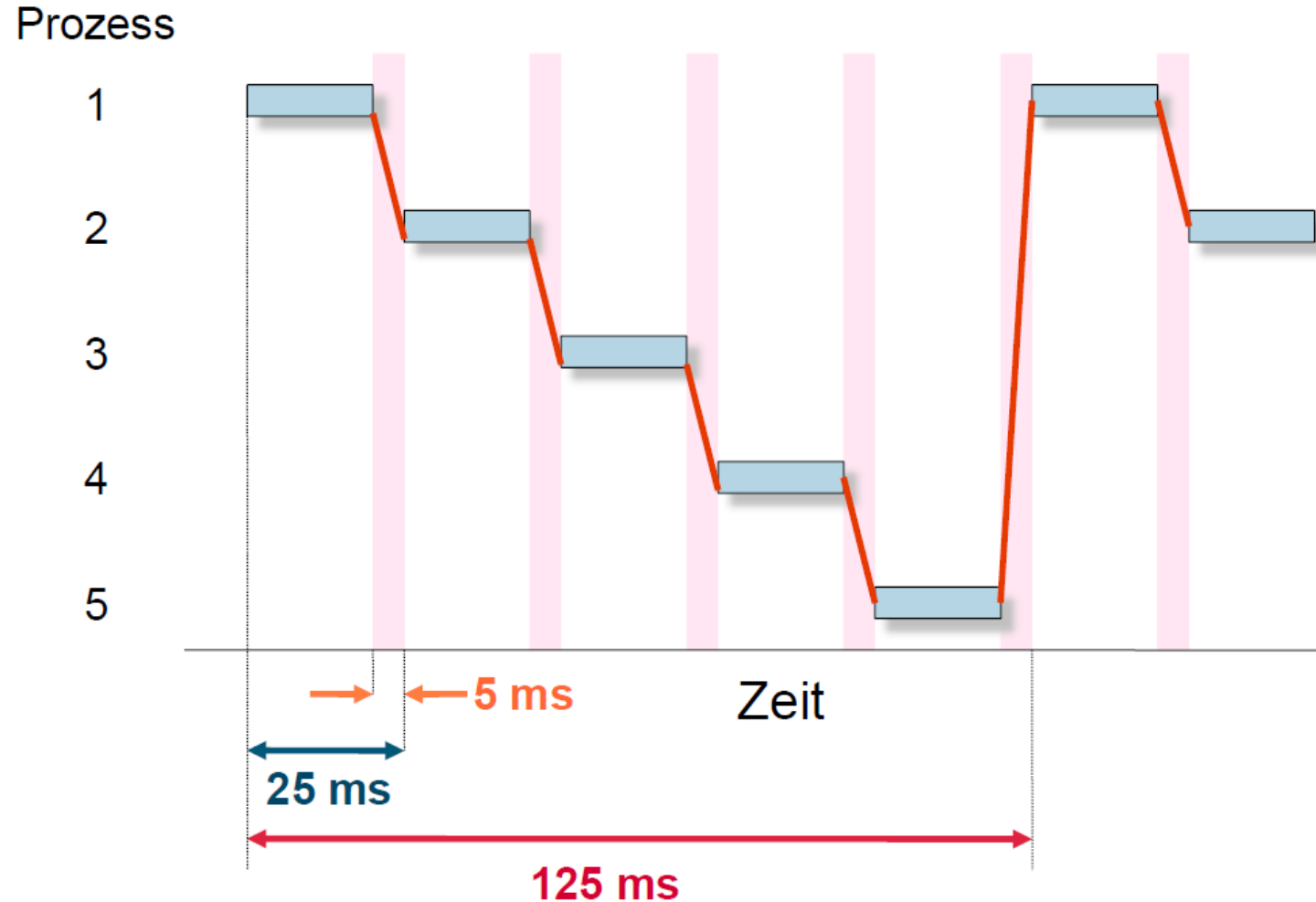
Sequentielle Verarbeitung

- › Wird ein Programm gestartet, werden alle enthaltenen Befehle sequentiell abgearbeitet
 - › Inklusive Sprünge durch Verzweigungen und Schleifen
- › D.h. dauert die Ausführung eines einzelnen Befehls oder z.B. einer Schleife sehr lange, kann die Anwendung in dieser Zeit keine weiteren Befehle abarbeiten
- › Eine parallele (gleichzeitige) Verarbeitung innerhalb eines Programms findet nicht statt

Multitasking

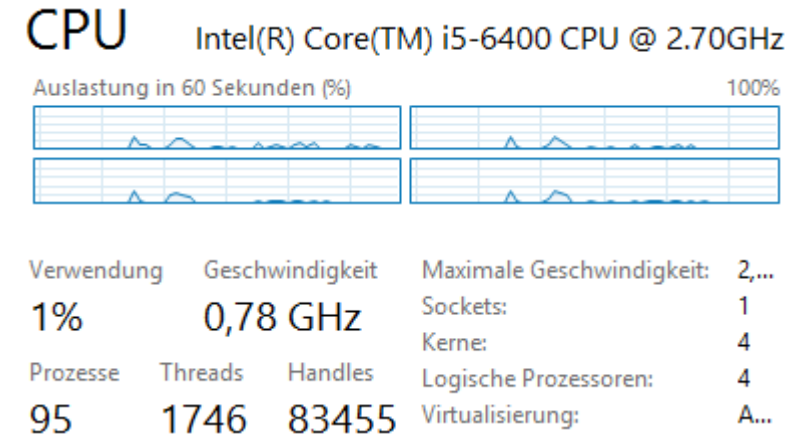
- › Ein Multitasking-Betriebssystem kann mehrere Anwendungen bzw. Prozesse parallel laufen lassen
- › Auch bei nur einem Prozessor können mehrere Anwendungen parallel ausgeführt werden, allerdings „fühlt“ sich das nur für den Benutzer so an
- › Das Betriebssystem teilt die Rechenzeit einer CPU in sogenannte Zeitscheiben bzw. Zeitfenster auf
- › Der Prozessor kann in einer Sekunde z.B. 20 Millisekunden (ms) für Prozess 1 einteilen, 20 ms für Prozess 2, 20 ms für Prozess 3, dann wieder Prozess 1, usw.

Zeitscheiben bzw. Round-Robin



Mehrprozessor- bzw. Mehrkernsystem

- › Hat ein System mehr als einen Prozessor, können mehrere Prozesse gleichzeitig ausgeführt werden



- › Die zur Verfügung stehende Zeit pro Prozessor wird nicht mehr aufteilt sondern steht einem Prozess komplett zur Verfügung
- › Eine einzelne Anwendung kann nicht ohne weiteres mehr als einen Prozessor verwenden

Parallelverarbeitung und Multitasking in Prozessen

- › Wird ein System mit mehreren Prozessoren verwendet, ist die Verwendung dieser für die Abarbeitung von Befehlen einer Anwendung wünschenswert
- › Auch schon bei einem Prozessor wäre es wünschenswert die Abarbeitung eines Code-Teils pausieren bzw. unterbrechen zu können, um andere Aufgaben der Anwendung auszuführen
 - › Beispiel: Bei Ausführung einer intensiven Berechnung weiterhin auf den Abbruchwunsch eines Benutzers reagieren zu können
- › Als Lösungsstrategie könnte Threading verwendet werden

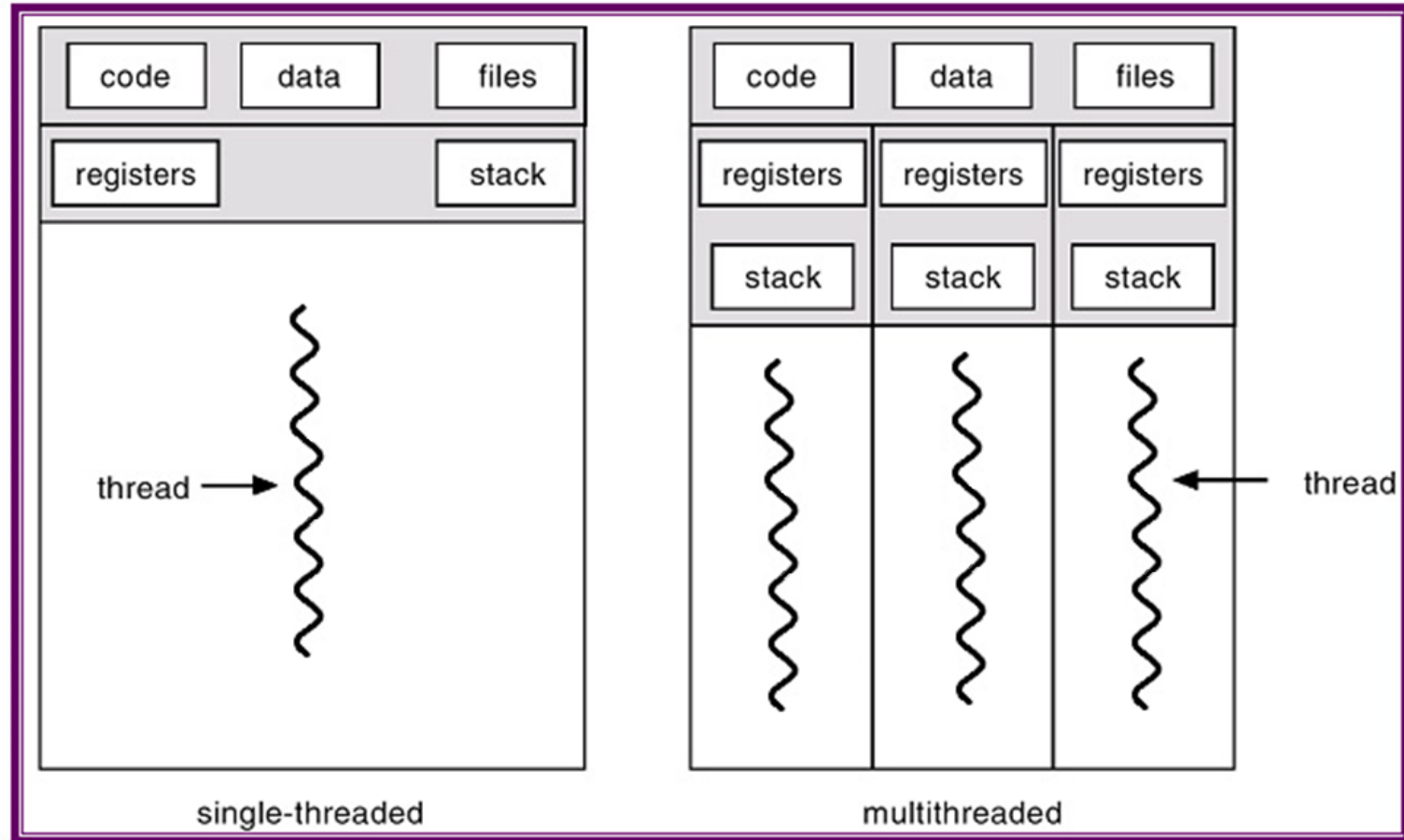
Thread

- › Aus dem englischen „Faden“ oder „Strang“
- › Ein Thread ist Teil eines Prozesses
- › Wird auch als leichtgewichtiger Prozess bezeichnet
- › Ist ein Ausführungsstrang bzw. Ausführungsreihenfolge in der Abarbeitung eines Programms
- › Threads werden durch das Betriebssystem gesteuert

Möglichkeiten von Threads

- › Bedeutet dass es innerhalb einer Anwendung verschiedene Code-Abschnitte gibt, zwischen welchen gewechselt werden kann oder welche parallel ausgeführt werden können

Singlethreaded vs. Multithreaded



Threading

- › Ziele bei der Verwendung von Threads:
 - › Parallele Verarbeitung von mehr als einem Befehl
 - › Möglichkeit zwischen der Ausführung unterschiedlicher Codeteile wechseln zu können
- › Eine Anwendung reagiert weiterhin auf Nutzerinteraktion obwohl ein Teil der Anwendung lange zur Ausführung dauert
- › Eine Anwendung kann durch nebenläufige Ausführung eine Aufgabe ggf. schneller verarbeiten
 - › Das ist jedoch kein Muss und auch nicht immer möglich!

Begriffe

- › Nebenläufigkeit
 - › Zwei oder mehr Befehle werden zur gleichen Zeit abgearbeitet
- › Asynchrone Verarbeitung
 - › Die Abarbeitung eines Befehls kann unterbrochen werden, um einen anderen zu verarbeiten
 - › Die Verarbeitung des ersten Befehls wird anschließend fortgesetzt
 - › Dies kann nebenläufig erfolgen, muss es aber nicht!!!

Vorteil von asynchroner Verarbeitung

- › Werden z.B. große Datenmengen (z.B. Dateien) verarbeitet kann durch asynchrone Verarbeitung sichergestellt werden, dass die Anwendung auch beim Einlesen der Dateien weiterhin reagiert
- › Erlaubt es z.B. eine Fortschrittsanzeige darzustellen

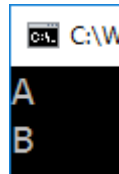
Nachteile von asynchroner Verarbeitung

- › Nicht jede Aufgabe ist (sinnvoll) parallelisierbar
- › Asynchrone bzw. parallele Verarbeitung ist immer mit gesondertem Aufwand verbunden und daher z.B. für sehr kleine Aufgaben nicht geeignet
- › Parallelverarbeitung kann ggf. sehr aufwändig werden und Fehler verursachen, welche unter Umständen nur sehr schwer zu finden sind

Determinismus

- › Bei einem deterministischen Algorithmus treten nur reproduzierbare Zustände auf
- › Bei gleicher Eingabe folgt auch immer die gleiche Ausgabe
- › Alle Zwischenergebnisse innerhalb des Algorithmus sind immer gleich
- › Die beiden Anweisungen führen immer zur gleichen Ausgabe

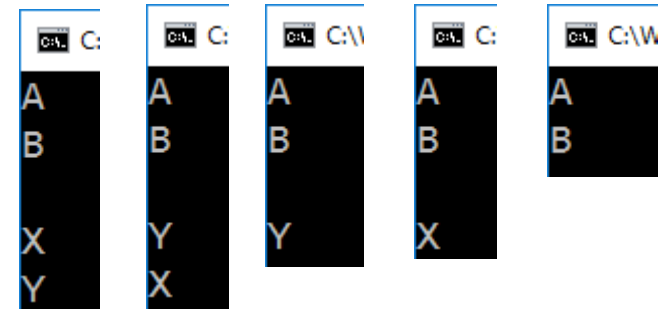
```
Console.WriteLine("A");  
Console.WriteLine("B");
```



Kein Determinismus mit Threads

- › Werden Threads verwendet, kann nicht vorhergesagt werden in welcher Reihenfolge die Threads gestartet und abgearbeitet werden
- › Die Ausgabe des Programms mit zwei Threads wird oft gleich aussehen, dies kann jedoch nicht garantiert werden!!!

```
Console.WriteLine("A");  
Console.WriteLine("B");  
  
Console.WriteLine();  
  
Task.Run(() => Console.WriteLine("X"));  
Task.Run(() => Console.WriteLine("Y"));
```



Gemeinsamer Zugriff auf Variablen aus Threads

- › Threads können Daten (z.B. Variablen) gemeinsam nutzen
- › Die gemeinsame Nutzung von Daten birgt jedoch einige Gefahren
- › Bedingt durch Hardware (CPUs, Speicher, Caches, Bus-Systeme) und das Betriebssystem

Gemeinsame Nutzung eines Zählers

```
private static int zähler;
```

0 Verweise

```
private static void Main(string[] args)
{
    Task task1 = Task.Run(ThreadMethod);
    Task task2 = Task.Run(ThreadMethod);

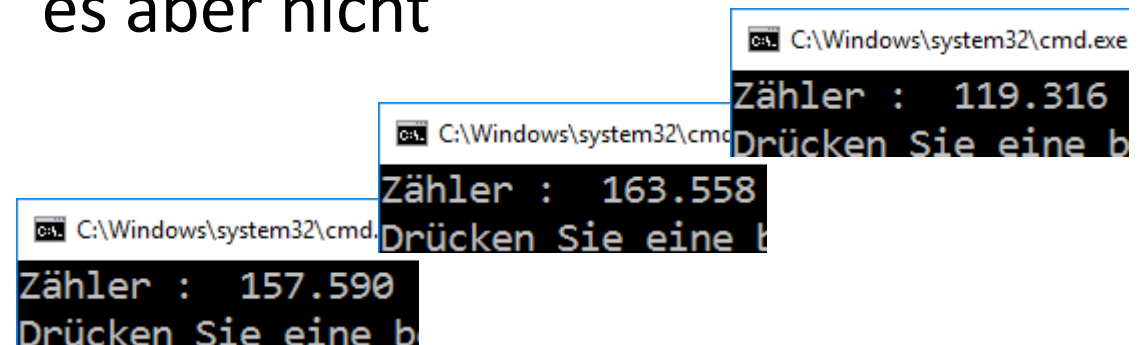
    Task.WaitAll(task1, task2);

    Console.WriteLine("Zähler : {0,8:N0}", zähler);
}
```

2 Verweise

```
private static void ThreadMethod()
{
    for (int i = 0; i < 100_000; i++)
    {
        zähler++;
    }
}
```

- › Die Variable zähler wird von beiden Threads verwendet
- › Jeder Thread erhöht die Variable 100.000-mal
- › Bei zwei Threads wird ein Ergebnis von 200.000 erwartet
- › Dies kann ggf. auch eintreten, muss es aber nicht



Threadsicherheit

- › Ein Objekt, das auch dann in einem gültigen Zustand bleibt, wenn mehrere Threads gleichzeitig auf dieselbe Ressource zugreifen
- › Bedeutet, dass mehrere Threads gleichzeitig dieselbe Methode desselben Objekts aufrufen können, ohne dass es zu Konflikten kommt
- › Auskunft über Threadsicherheit der Klassen und Methoden von .NET sind in der MSDN zu finden

Threadsicherheit

- › Um eine threadsichere Nutzung von gemeinsamen Ressourcen sicherzustellen ist oft viel Aufwand notwendig
- › Frameworks wie .NET stellen Entwicklern eine Vielzahl von Hilfsmitteln in Form von Klassen und Methoden zur Verfügung
 - › Mutex, Semaphore, kritische Bereiche, usw.
- › Die Verwendung der Klassen für eine threadsichere Anwendung ist ggf. nicht trivial und zeitaufwendig
 - › Im schlimmsten Fall ist die vermeintliche parallele Anwendung langsamer als die serielle
- › Werden die Klassen zur Synchronisierung nicht ordnungsgemäß verwendet, kann es zu sog. Deadlocks kommen und die Anwendung reagiert nicht mehr