

Threads

Teil 3

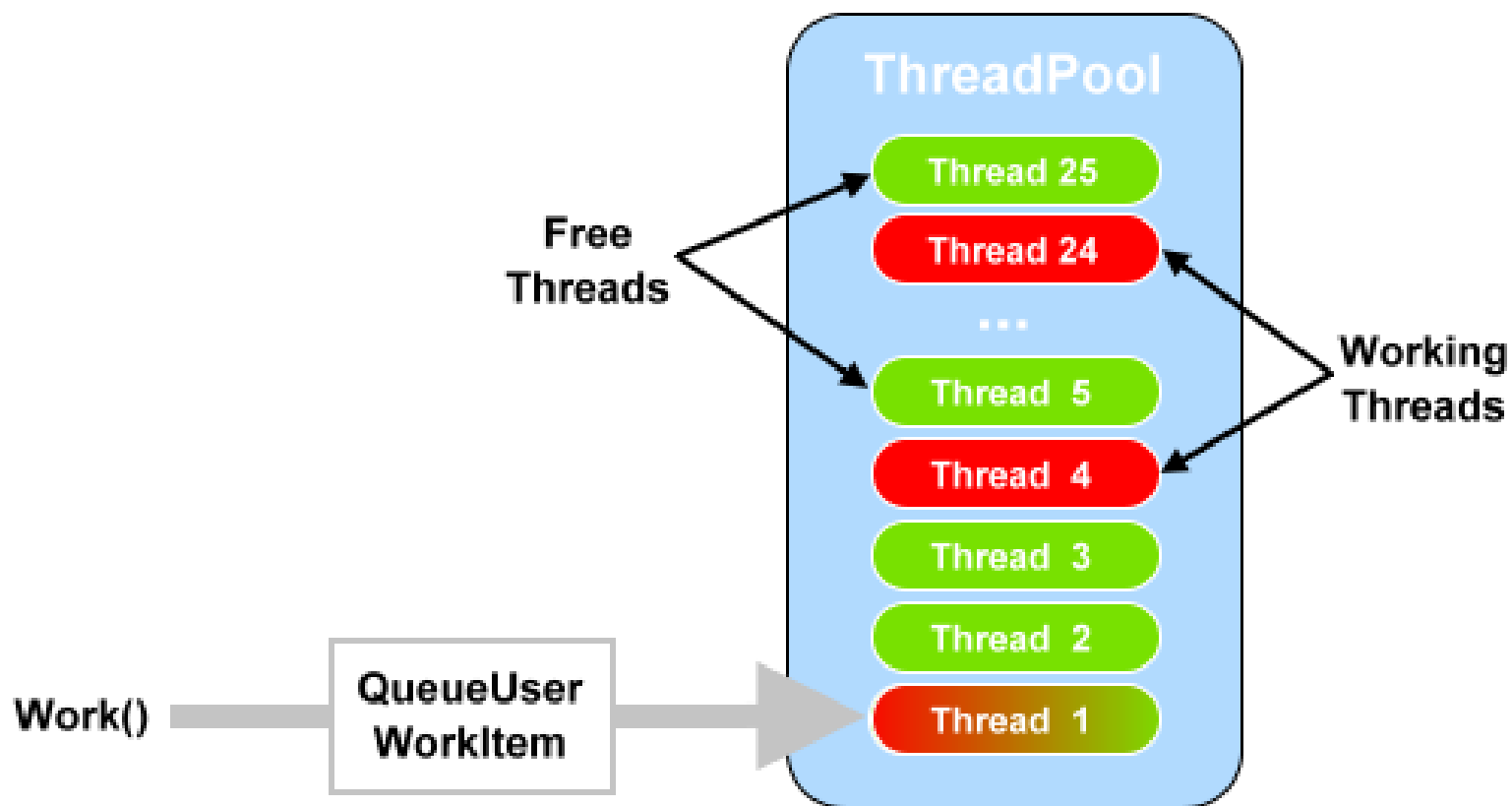
Ausführung synchroner Methoden

- der aufrufende Thread wird blockiert, solange der Thread der aufgerufenen Methode noch aktiv ist.
- am Ende gibt die Methode ihre Ergebnisse an den Aufrufer (engl. Caller) zurück.
- Der Caller erhält sicher die Rückgabewerte, sofern der Code fehlerfrei ist.
- Problem: aufrufender Thread ist solange blockiert, bis die aufgerufene Methode abgearbeitet ist.

Ausführung asynchroner Methoden

- Einsatz mehrerer Threads: Anwendungen können mehrere Arbeitsschritte parallel bearbeiten
- der aufrufende Thread wird nicht blockiert und kann sich anderen Dingen widmen
- Aufrufer muss Rückgabedaten erhalten
- Mechanismen von .NET für asynchrone Methoden:
 - ThreadPool
 - BackgroundWorker
 - Callback-Methoden

ThreadPool



ThreadPool

- Thread erstellen ohne die Klasse Thread zu benutzen
- In .NET gibt es genau einen Threadpool pro Prozess für Hintergrundthreads
- Standardgröße: 25 Arbeitsthreads pro verfügbarem Prozessor und 1000 I/O-Abschlussthreads.
- aufrufender Thread führt asynchron andere Aufgaben aus, während die aufgerufenen Threads im Pool sich um die in der Warteschlange befindlichen Arbeiten kümmern.
- Threads reihen sich nach Beendigung ihrer Arbeit wieder in die Schlange der wartenden Threads ein
- Nach Beendigung führt ein Arbeitsthread aus dem Threadpool die entsprechende Rückruffunktion (Callback) aus.

```
public class Rechnung {
    private int n;
    private int ergebnis;
    private ManualResetEvent doneEvent;
    public Rechnung(int n, ManualResetEvent doneEvent) {
        this.n = n;
        this.doneEvent = doneEvent;
    }
}
```

// Wrapper Methode für den Threadpool.

```
public void ThreadPoolCallback(Object threadContext) {
    int threadIndex = (int)threadContext;
    Console.WriteLine("Thread {0} beginnt mit Berechnung...", threadIndex);
    ergebnis = Calculate(n);
    Console.WriteLine("Thread {0} fertig...", threadIndex);
    // Benachrichtigung, dass die Berechnung abgeschlossen ist.
    doneEvent.Set();
}
```

// Rechnung

```
public int Calculate(int n) {
    int summe = 0;
    for (int i = 0; i < n; i++) { summe = summe + i; }
    return summe;
} }
```

```
public void Start() {
```

```
// N, Lower und Upper definieren und Werte zuweisen, Zufallsgen. r anlegen
```

```
// Für jedes Rechnung-Objekt wird ein Ereignis verwendet
```

```
ManualResetEvent[] doneEvents = new ManualResetEvent[N];
```

```
Rechnung[] Array = new Rechnung[N];
```

```
Random r = new Random();
```

```
// Konfiguriere und starte den ThreadPool
```

```
Console.WriteLine("Starte {0} Aufgaben...", N);
```

```
for (int i = 0; i < N; i++) {
```

```
    doneEvents[i] = new ManualResetEvent(false);
```

```
    Rechnung f = new Rechnung(r.Next(Lower, Upper), doneEvents[i]);
```

```
    Array[i] = f;
```

```
// Übergebe die Aufgabe an den Pool
```

```
    ThreadPool.QueueUserWorkItem(new WaitCallback(f.ThreadPoolCallback), i);
```

```
}
```

```
// Warte darauf, dass alle Threads aus dem Pool ihre Arbeit abgeschlossen haben
```

```
WaitHandle.WaitAll(doneEvents);
```

```
Console.WriteLine("Alle Berechnungen abgeschlossen.");
```

```
// Zeige die Ergebnisse an
```

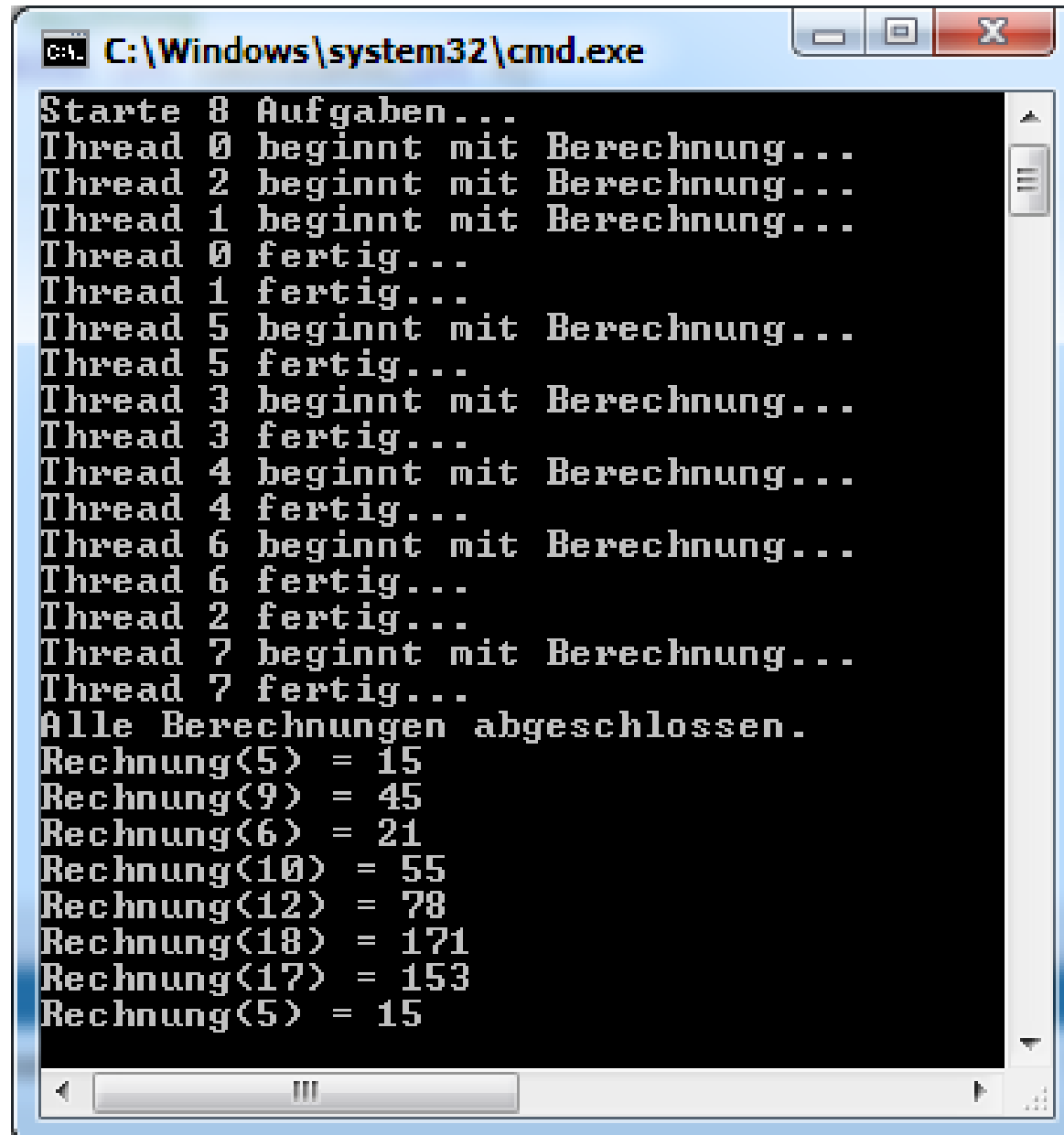
```
for (int i = 0; i < N; i++) {
```

```
    Rechnung f = Array[i];
```

```
    Console.WriteLine("Rechnung({0}) = {1}", f.N, f.Ergebnis);
```

```
}
```

ThreadPool



```
C:\Windows\system32\cmd.exe

Starte 8 Aufgaben...
Thread 0 beginnt mit Berechnung...
Thread 2 beginnt mit Berechnung...
Thread 1 beginnt mit Berechnung...
Thread 0 fertig...
Thread 1 fertig...
Thread 5 beginnt mit Berechnung...
Thread 5 fertig...
Thread 3 beginnt mit Berechnung...
Thread 3 fertig...
Thread 4 beginnt mit Berechnung...
Thread 4 fertig...
Thread 6 beginnt mit Berechnung...
Thread 6 fertig...
Thread 2 fertig...
Thread 7 beginnt mit Berechnung...
Thread 7 fertig...
Alle Berechnungen abgeschlossen.
Rechnung(5) = 15
Rechnung(9) = 45
Rechnung(6) = 21
Rechnung(10) = 55
Rechnung(12) = 78
Rechnung(18) = 171
Rechnung(17) = 153
Rechnung(5) = 15
```