Verteilte Systeme

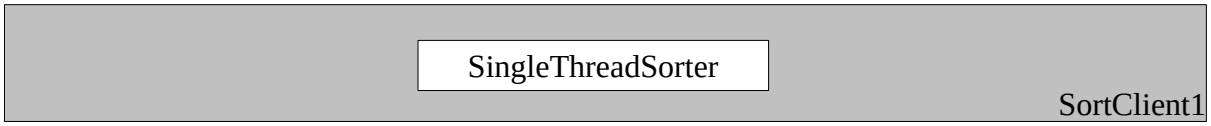
Übung D

Bearbeitungszeit 2 Wochen

Die Übung adressiert die Verteilung von Algorithmen in Clustern und Supercomputern mittels Rekursion über Prozesse und Threads. Zur Demonstration wird ein MergeSort-Algorithmus verwendet und erweitert. Die Messungen müssen auf Mehrkernsystemen ausgeführt werden um sinnvolle Ergebnisse zu liefern, beachtet jedoch dabei ob es sich bei Euren Computern um echte Mehrkernsysteme, oder nur um virtuelle Barrel-Technologie bzw. Hyperthreading handelt! Brauchbare Quellen sind dazu:

- Merge-Sort Verfahren: http://en.wikipedia.org/wiki/Merge_sort
- EBNF-Syntaxbeschreibungen: http://en.wikipedia.org/wiki/EBNF

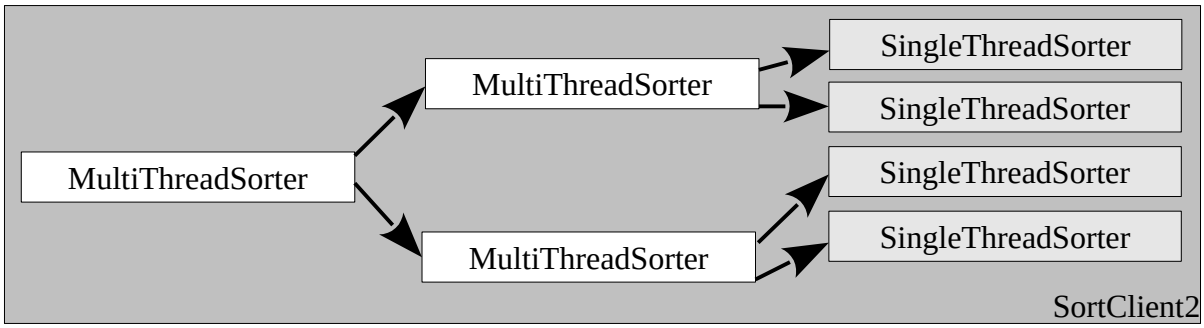
Aufgabe 1: Einfache Rekursion



Vorarbeit: Verwendet einen Archivmanager, einen Text-Editor sowie Copy&Paste um aus der Datei `goethe-faust.zip` eine Text-Datei `goethe-faust-x50.txt` in 50-facher Länge zu replizieren. Eine solche wird benötigt um für die kommenden Sortieroperationen genügend Daten verfügbar zu haben.

Entfernt „Skeleton“ aus allen Klassennamen in denen diese Namen-Endung vorkommt. Macht Euch mit dem Interface **MergeSorter** sowie den Klassen **SingleThreadSorter**, **SortClient1**, sowie deren abstrakter Superklasse **SortClient** vertraut. Bringt die Applikation **SortClient1** zum Laufen, und messt aus wie lange es dauert die obige Datei nach Wörtern zu sortieren.

Aufgabe 2: Rekursion über multiple Threads eines Prozesses

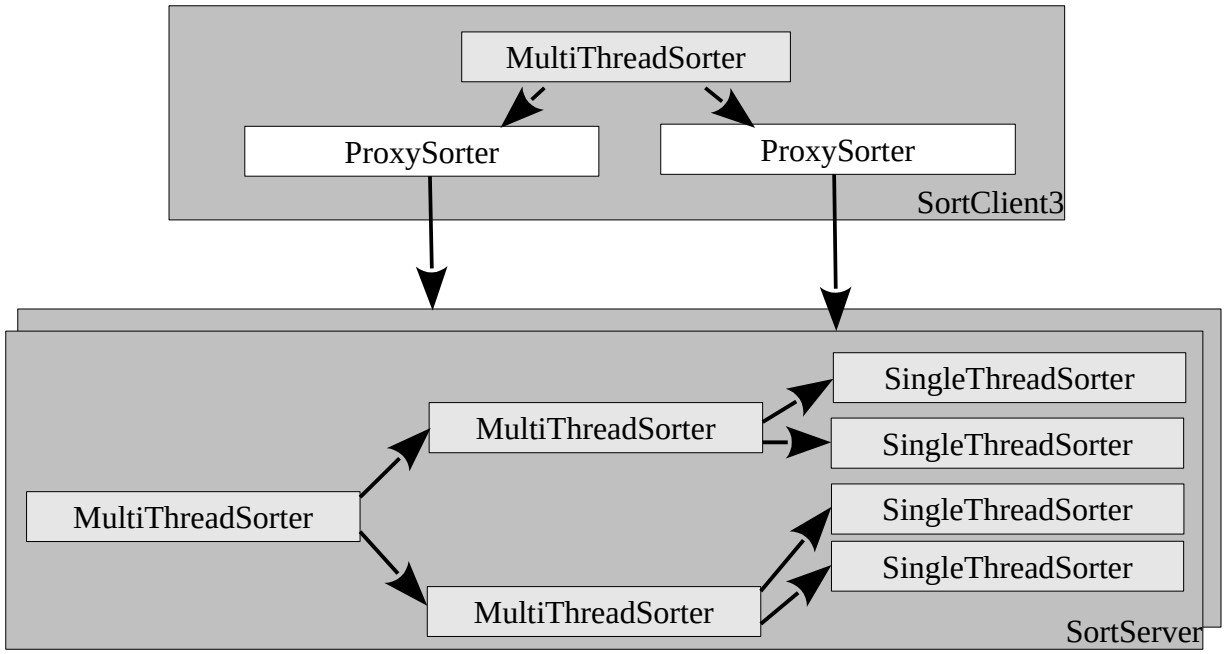


Komplettiert die Implementierung der Klasse **MultiThreadSorter** wie im Source-Code unter TODO beschrieben. Kopiert die Klasse **SortClient1** nun nach **SortClient2**, und schreibt sie so um dass sie `MultiThreadSorter.newInstance()` zur Erzeugung der

Sortierers-Instanzen verwendet.

Testet Eure Implementierung auf korrekte Funktion, und messt aus wie lange es dauert die Datei `goethe-faust-x50.txt` mittels **SortClient2** zu sortieren. Vergleicht die Laufzeit mit Aufgabe 1.

Aufgabe 3: Rekursion über multiple Prozesse und Threads



Analysiert die Klassen **ProxySorter** und **SortServer**, vor allem die darin beschriebene EBNF-Protokollsyntax. Komplettiert die Implementierung dieser beiden Klassen wie im Source-Code unter TODO beschrieben.

Kopiert die Klasse **SortClient1** nach **SortClient3**, und schreibt sie so um dass eine beliebige strikt positive Zahl von Socket-Adressen (z.B. „host:8002“) externer SortServer als zusätzliche Parameter übernommen werden – ihr könnt zum Parsen einzelner Socket-Adressen den Ausdruck `InetAddresses.toSocketAddress(text)` verwenden. Verwendet zudem den Aufruf `ProxySorter.newInstance(socketAddresses)` zum Erzeugen des Sortierers in der main()-Methode.

Startet zum Testen der Funktion Eurer Implementierung zuerst den Spezialfall eines einzelnen **SortServer** in Verbindung mit einem **SortClient3**. Sobald Eure Implementierung fehlerfrei läuft, startet vor **SortClient3** je einen **SortServer** auf bis zu 8 separaten Maschinen (i.e. bis zu 8 Knoten eines Multi-Computer-Clusters, z.B. unseres HTW-Labors oder Eures heimischen WLAN-Netzes), und konfiguriert entsprechend auf einer der Maschinen eine Instanz von SortClient3. Messt wie lange es nun dauert die Datei `goethe-faust-x50.txt` mittels dieses Clients zu sortieren. Vergleicht die Laufzeit mit Aufgabe 1 und 2.