Prof. Dr. Th. Letschert

THM TECHNISCHE HOCHSCHULE MITTELHESSEN
MNI Mathematik, Naturwissenschaften und Informatik

NVP – Nebenläufige und Verteilte Programme

Aufgabenblatt 7

Aufgabe 1

Mit Java 6 hat Sun das *package* `com.sun.net.httpserver`[1] zur Java-API hinzugefügt. Es bietet einige Klassen mit deren Hilfe ein HTTP-Server mit wenig Mühe erstellt werden kann. Leider wird das *package* von Oracle ein wenig stiefmütterlich behandelt und es finden sich auch nur einige wenige Beispiele oder Tutorials im Netz.

Einfache eingebettete Webserver können damit aber gut und schnell implementiert werden. Beispielsweise ein Webserver der auf die Faktorisierung spezialisiert ist:

```
package blatt_07.aufgabe_01.server

import java.net.InetSocketAddress
import com.sun.net.httpserver.{HttpExchange, HttpHandler, HttpServer}
import java.util.concurrent.{Executors, ExecutorService}

object Factorizer {

  import scala.util.{Failure, Success, Try}

  def factors(n: Long): Try[List[Long]] = ....

  def factors(x: String): String = try {
    Factorizer.factors(x.toLong) match {
      case Success(l) => l.toString
      case Failure(e) => e.toString
    }
  } catch {
    case e: NumberFormatException => e.toString()
  }

}

/* Html-Code der vom Server ausgeliefert wird.
 * Der Sever liefert eine Html-Seite zum Faktorisieren.
 * Diese Seite nutzt Ajax um mit dem Server (asynchron) zu kommunizieren.
 */
object HtmlCode {
  val ajax =
"""
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8"/>
   <title>Let's go Ajax</title>
   <script type="text/javascript">
     var xhr = new XMLHttpRequest();
```

[1] https://docs.oracle.com/javase/8/docs/jre/api/net/httpserver/spec/com/sun/net/httpserver/package-summary.html

```
    function sendRequest(request) {
      xhr.open( "get", "http://127.0.0.1:4711/factorize?"+request, true);
      xhr.onreadystatechange = handleResponse;
      xhr.send(null);
    }
    function handleResponse() {
      if (xhr.readyState == 4) {
        document.getElementById("ausgabe").innerHTML = xhr.responseText;
      }
    }
   </script>
</head>
<body>
<h1>Ajax Request</h1>
<form>
   <p>Hier Ihre Zahl: <input name="ZAHL" id="ZAHL" size="10"></p>
   </p>
</form>
<p>
   <button id="rechne">Faktorisiere</button>
</p>
<p>
<div id="ausgabe" style="width:150px; height:50px; background:lightgrey;
    font-size:small; font:Courier"></div>
</p>
</body>
  <script type="text/javascript">
    document.getElementById("rechne").onclick = function() {
      var z = document.getElementById("ZAHL").value
      sendRequest("ZAHL="+z)
    };
  </script>
</html>
"""
}

/* Der Sever
*/
object FactorizingHttpServer {
  val numberOfCores = Runtime.getRuntime().availableProcessors()

  def main(args: Array[String]) {
    val server = HttpServer.create(new InetSocketAddress(4711), 0)
    server.createContext("/", RootHandler)
    server.createContext("/factorize", FactorHandler)

    // Ein Executor fuehrt die Handler aus
    val executor: ExecutorService = Executors.newFixedThreadPool(numberOfCores*5)
    server.setExecutor(executor)

    server.start()
    println("Hit return to stop server")

    System.in.read()
    executor.shutdown()
    server.stop(0)
  }
}

/* Der Root-Handler liefert die Html-Seite aus.
*/
object RootHandler extends HttpHandler {
```

```
  def handle(httpExchange: HttpExchange) {
    val uri = httpExchange.getRequestURI()
    val path = uri.getPath
    if (path == "/ajax.html") {
      httpExchange.sendResponseHeaders(200, HtmlCode.ajax.length)
      httpExchange.getResponseHeaders().set("Content-Type","text/html; charset=utf-8");
      val os = httpExchange.getResponseBody
      os.write(HtmlCode.ajax.getBytes)
      os.close()
    } else {
      val response = "404 (Not Found)\n"
      httpExchange.sendResponseHeaders(404, response.length())
      val os = httpExchange.getResponseBody()
      os.write(response.getBytes())
      os.close()
    }
  }
}

/* Ein Handler fuer Faktorisierungs-Anfragen
*/
object FactorHandler extends HttpHandler {

  def editQuery(query: String): Map[String, String] =
    query.split("&").
      map(_.split("=")).
      foldLeft( Map[String, String]() )(
        (acc: Map[String, String],
         p: Array[String]) => acc + (p(0) -> p(1))
        )

  def handle(httpExchange: HttpExchange): Unit = {
    val uri = httpExchange.getRequestURI()
    val query = uri.getQuery()
    if (query != null) {
      val queryParams = editQuery(query)
      val factors = Factorizer.factors(queryParams("ZAHL"))
      httpExchange.sendResponseHeaders(200, factors.getBytes().length)
      val os = httpExchange.getResponseBody()
      os.write(factors.getBytes())
      os.close()
    }
  }
}
```

Diese minimalistische Version kann natürlich noch ausgeschmückt werden: Im Client CSS zur Verschönerung, jQuery zur einfacheren Implementierung, ein *document root* damit Dateien geladen werden können, ... (Achtung der HTML-Code muss von der IP-Adresse geladen werden, an die die Anfrage gesendet wird, und *"127.0.0.1"* ist dabei nicht das Gleiche wie *"localhost"*)

1. Realisieren nach dem Muster dieses Beispiels einen Web-Server der Faktorisierungsanfragen von einem Client (Browser) annimmt und ausführt.
2. Warum ist Ajax keine (!) geeignete Technologie, wenn es möglich sein soll, laufende Faktorisierungen vom Client aus zu unterbrechen?
3. Modifizieren / erweitern Sie diese Anwendung derart, dass ein Cache zum Einsatz kommt, der verhindert, dass die gleiche Faktorisierung mehrfach ausgeführt wird. (Korrekte Synchronisation!) Achten Sie darauf, dass auch eine bereits begonnene aber noch laufende Faktorisierung verhindert, dass die gleiche noch einmal gestartet wird. (Hinweis: Cachen Sie Futures.)

Aufgabe 2

Gegeben sei eine Liste von Zahlen. Das *Problem der maximalen Segment-Summe* (*Maximum Segment Sum, MSS*) besteht darin, die maximale Summe eines zusammenhängenden Segments der Liste zu bestimmen. Beispielsweise ist 21 die MSS von $(3, 5, 10, -5, -30, 5, 7, 2, -3, 10, -7, 5)$. Es ist die Summe des Segments $(5, 7, 2, -3, 10)$. Das Problem tritt in vielen praktischen Anwendungen auf, beispielsweise in der Bioinformatik bei der Analyse von Protein- oder DNA-Sequenzen.

Die maximale Sequenzsumme ist einfach das Maximum der Summen aller Sub-Sequenzen in einer Sequenz. Eine Sub-Sequenz einer Sequenz ist dabei ein Präfix von einem Suffix. Das ergibt folgende *brute force / exhaustive search* Lösung des MSS-Problems:

```
object MSS_BruteForce {

  def suffix[A](l: List[A]): List[List[A]] = l match {
    case Nil => Nil
    case _ => l :: suffix(l.tail)
  }

  def prefix[A](l: List[A]): List[List[A]] = l match {
    case Nil => Nil
    case head::tail => Nil :: prefix(l.tail) map(head :: _)
  }

  def mms(l: List[Int]): Int = prefix(l).flatMap(suffix(_)).map(_.sum).max
}
```

Ein Teile-und-Herrsche-Algorithmus[2] zur Lösung des MSS-Problems

- teilt die Liste in der Mitte und berechnet dann
- (1) die maximale Sequenzsumme des linken Teils,
- (2) die maximale Sequenzsumme des rechten Teils und
- (3) das Maximum der Summe der die Mitte überlappenden Sequenzen.
- Das Maximum dieser drei Werte ist das Ergebnis.

Die maximale Sequenz, die die Mitte überlappt, besteht aus einem Suffix des linken Teils und einem Prefix des rechten Teils. Man macht sich schnell klar, dass der maximale Suffix der linken Seite verknüpft mit dem maximalen Prefix der rechten Seite die maximale Sequenz ergibt, die die Mitte überlappt. Insgesamt:

```
object MSS_DivideAndConquer {

  def suffix[A](l: List[A]): List[List[A]] = ...
  def prefix[A](l: List[A]): List[List[A]] = ...

  def max (x: Int, y: Int): Int = if (x>y) x else y
  def max (x: Int, y: Int, z: Int): Int = max(max(x, y), z)

  def midSum(l_l: List[Int], l_r: List[Int]): Int =
    suffix(l_l).map(_.sum).max +
    prefix(l_r).map(_.sum).max

  def mms(l: List[Int]): Int =
    if (l.size == 1) max(l(0), 0)
    else {
```

[2] vermutlich ursprünglich veröffentlicht von Dijkstra und Feijen in *Een methode van programmeren*, Academic Service, 's Gravenhage, 1984. Der Algrorithmus hier folgt John Bentley: *Programming Pearls: Algorithm Design Techniques*, CACM 27, 9 online http://www.akira.ruc.dk/ keld/teaching/algoritmedesign_f03/Artikler/05/Bentley84.pdf

```
    val mid = l.size/2
    max(
      mms(l.slice(0, mid)),
      mms(l.slice(mid, l.size)),
      midSum(l.slice(0, mid), l.slice(mid, l.size))
    )
  }
}
```

Implementieren Sie eine parallelisierte Version der Teile-und-Herrsche-Version des Algorithmus' mit Hilfe des *Fork-Join-Frameworks*.