Fortgeschrittene funktionale Programmierung in Haskell

Übungszettel 7

Aufgabe 7.1:

In dieser Aufgabe sollen Sie einen einfachen, thread-basierten Server für kleinere Berechnungen programmieren, analog zu einer beliebten Aufgabe aus dem Modul `Betriebssysteme`.
Eine Vorlage für die Kommunikation über das Netzwerk, die Verbindungen von `telnet`-Clients entgegen nimmt, finden Sie im Repository `uebungen`.

Ihr Server soll gleichzeitig mehrere Clients bedienen können. Zusätzlich soll jeder dieser Clients wiederum in der Lage sein, beim Server mehrere Jobs in Auftrag zu geben, die vom Server simultan bearbeitet werden. Der Server sollte also nach dem Start eines Jobs sofort wieder Kommandos entgegennehmen, z.B. weitere Jobs starten können. Der Client kann sich (anhand einer vom Server ausgegeben Job-Nummer) jederzeit über den Bearbeitungsstand der Jobs erkundigen, das Ergebnis des Jobs erfragen oder den Job abbrechen.

Ihr Server soll einfache Ausdrücke der Form „Zahl Operator Zahl“ entgegennehmen und den entsprechenden Wert berechnen (z.B. `7 + 3`). Verzögern Sie die Bearbeitung dieser Berechnung um einige Sekunden (Wie lässt sich das erreichen, ohne dass *Lazy Evaluation* Ihnen einen Strich durch die Rechnung macht?).

Ihr Server soll über mindestens folgende Kommandos verfügen:

- Das Kommando `job` startet einen Berechnungsjob unter Angabe des Ausdrucks (s.o). Der Server weist dem Job eine Jobnummer zu und gibt diese an den Client als Zeichenkette zurück, z.B.

  ```
  job 6 / 3
    created job no. 2
  job 4.5 + 3.1
    created job no. 3
  ```

  Die Antworten des Servers sollen jeweils eingerückt sein (zwei Leerzeichen zu Beginn).

- Das Kommando `ready` erkundigt sich beim Server, ob der Job mit der angegebenen Jobnummer (wie bei der Job-Erzeugung vom Server angegeben) bereits beendet wurde, z.B.

  ```
  job 8 / 4
    created job no. 4
  ready 4
    job 4 is not ready
  ready 4
    job 4 is ready
  ```

- Das Kommando `get` holt die Ergebnisse des Jobs mit der angegebenen Jobnummer vom Server. Ist der Job noch nicht beendet, so wird auf die Fertigstellung blockierend gewartet (der Server soll in dieser Zeit nicht von diesem Client ansprechbar sein).

Ein Beispiel, bei dem `get` nicht warten muss, weil der Job bereits fertiggestellt wurde:

```
job 1 + 2
  created job no. 8
ready 8
  job 8 is not ready
ready 8
  job 8 is ready
get 8
  result: 3
```

- Das Kommando `list` listet alle Jobs des Clients auf. Für beendete Jobs wird das Ergebnis angezeigt (diese werden hierbei nicht aus der Jobliste entfernt).:

```
list
  job 3 with task 8 / 4 produced 2
  job 4 with task 1 + 2 is running
```

- Das Kommando `quit` beendet die Kommunikation mit dem Server. Der Client (hier `telnet`) endet beim Schließen des Kommunikationskanals im Server automatisch mit der Meldung „Connection closed by foreign host“.

Die Kommandos sollten entsprechende Fehlermeldungen an den Client liefern. Insbesondere sollen Jobs, deren Ergebnisse bereits abgefragt wurden (`get`) aus der Jobliste gelöscht werden, also nicht mehr unter ihrer Jobnummer ansprechbar sein. Beispiele für Fehlermeldungen:

```
job falsche Argumente
  created job no. 0
get 0
  result: invalid task format
job 2 # 4
  created job no. 1
get 1
  result: computation failed
get 2
  job 2 not found
get 0
  job 0 not found
get zuviele Argumente
  erroneous get command: get zuviele Argumente
falsches Kommando
  invalid command: falsches Kommando
```