Übungszettel 4

Allgemein:
Abgabe via OLAT Upload bis Montag vor der nächsten Übung um 08:00! Für dieses Übungsblatt Abgabe bis 18.1.2016!
Upload muss aus einem File mit Beschreibung der Lösung (pdf oder txt) **und** einem weiteren Archiv das Source-Code enthält (zip oder tar.gz, Eclipse Project exportieren!) bestehen. Punktabzug bei falschem Dateiformat oder mehr oder weniger als 2 Files (kein docx, tex, rar, 7z, Ohne Endung, …) bestehen! Zwei gültige Files z.B. name-u1.txt und name-u1.zip

Nützlicher Link: http://jcip.net.s3-website-us-east-1.amazonaws.com/listings.html

1) Cancellation: (40%)

In dieser Aufgabe geht es um die Implementierung von Mechanismen die in der Vorlesung besprochen wurden: Interrupt cancellation, Interrupt restore und Non-interruptible blocking

a) Schreiben Sie ein Programm, in dem 4 Threads miteinander kommunizieren (message passing, queues, .. die Kommunikation muss Thread-save sein!) mit entsprechender Konsolenausgabe:
    a) Thread 1 erstellt eine Zufallszahl [0-41] fuer Thread 2
    b) Thread 2 überprüft ob die Zahl eine gerade Zahl ist und gibt alle geraden Zahlen weiter
    c) Thread 3 schreibt die Zahl in einen synchronus Socket der zur kommunikation mit Thread 4 verwendet wird (zusätzlich kann eine beliebige Datenmenge in den Socket geschrieben werden um die Kommunikationszeit zu erhöhen).
b) Arbeiten Sie mit Hilfe von sleep() um die Ausführung zu Verlangsamen.
c) Es soll möglich sein einen kontrollierten Shutdown der 4 Threads zu initiieren.
    a) Thread 1 kann beliebig gestoppt werden (Thread-save!)
    b) Thread 2 soll eine Poison Pill von Thread 1 erhalten (wird beim beenden von Thread 1 generiert).
    c) Main soll kontrollieren wann Thread 1 und 2 gestoppt wurden und dann Thread 3 beenden (Vorgabe: die Abarbeitung des Sockets dauert zu lange und muss händisch abgebrochen werden)
    d) Thread 4 soll mittels Interrupt mitgeteilt werden zu stoppen und den Interrupt Status restoren fuer den fall das der Code Innerhalb eines anderen Threads ausgeführt werden sollte.
d) Dokumentieren Sie die 4 Threads bezüglich ihrer Cancellation Policies und ihrer Datenaustauschsroutinen. Ein Benutzer sollte anhand dieser Dokumentation fähig sein die main() zu Ihren Threads zu schreiben, ohne den Source der 4 Threads zu kennen! Alle public Funktionen muessen entsprechend Dokumentiert sein. Bitte kopieren Sie diese Java-docu auch in die “Beschreibung der Lösung”!

2) LinkedBlockingDeque Workers: (30%)

Schreiben Sie ein Programm das Rechenzentren und abzuarbeitende Jobs simuliert.

a) Jedes Rechenzentrum verfügt über eine LinkedBlockingDeque in der Jobs hinterlegt werden können (entspricht einem Lokalem Resource Manager wie z.b. https://en.wikipedia.org/wiki/TORQUE).
b) Jeder Job hat eine gewisse Menge von Rechenarbeit, und jedes Rechenzentrum hat eine Rechenleistung. Die Rechenzeit ergibt sich dann aus (Rechenarbeit / Rechenleistung). Jeder Job mit einem Geldbetrag dotiert, welcher nach Abarbeitung dem jeweiligen Rechenzentrum gutgeschrieben wird.
c) Jedes Rechenzentrum kann, wenn es selbst keine Jobs mehr in der Queue hat, Jobs von den Queue-Enden der anderen Rechenzentren stehlen. Die Rechenzentren **versuchen** immer den Job mit dem besten (Preis / Rechenleistung) Verhältnis zu stehlen.
d) Erstellen Sie ein Hauptprogramm das 3 verschieden schnelle Rechenzentren erstellt und über einen Jobgenerator verfügt, der allen Rechenzentren in unterschiedlichen Zeitabständen Jobs mit variierendem Rechenaufwand und unterschiedlichen Geldbeträgen schickt (in die Queue legt).
e) Das Programm soll ausgeben wenn ein Rechenzentrum Jobs abgearbeitet hat, wie viel es eingenommen hat und wenn es Jobs von anderen Rechenzentren stiehlt.

3) FutureTask Parallelisation: (30%)

Implementieren Sie mit hilfe einer FutureTask Klasse (ähnlich Preloader.java von der JCIP Seite) ein Programm das Pi berechnet (siehe Blatt 1, Aufgabe 2 für Informationen zur Berechnung).

a) Die Klasse CalculatePartOfPi soll mittels FutureTask einen Bereich der Pi-Summe berechnen können. Der Zahlenbereich soll im Konstruktor übergeben werden.
b) Das Hauptprogramm soll eine konfigurierbare Anzahl von CalculatePartOfPi Objekten anlegen und diese anschliessend unmittelbar hintereinander starten.
c) Summieren Sie die Einzelergebnisse und kontrollieren Sie das Endergebnis. Der Fehler und das Ergebnis sollen ausgegeben werden.
d) Die Zeit bis zum Erreichen des Endergebnisses soll gemessen werden für 1, 2, 4, 80, 160, 320,1024 und 2048 CalculatePartOfPi Objekte (mit und ohne der Objekt Instanzierungszeit).
e) Welche Anwendungsgebiete eigenen sich besonders für FutureTasks? Nennen Sie 2 Beispiele und erläutern Sie die Vorteile die genau bei diesen Beispiele ausschlaggebend sind.