3ª Lista de Exercícios de Paradigmas de Linguagens Computacionais

Professor: Fernando Castor

Monitores:

Ciência da computação: Eric Lucena Palmeira Silva (elps), João Gabriel Santiago Maurício de Abreu (jgsma), Marcos Paulo Barros Barreto (mpbb), Tulio Paulo Lages da Silva (tpls),Victor Félix Pimenta (vfp).

CIn-UFPE – 2014.1
Disponível desde: 05/07/2014
Entrega: 17/07/2014

A lista deverá ser respondida em dupla. A falha em entregar a lista até a data estipulada implicará na **perda de 0,25 ponto** na média da disciplina para os membros da dupla. Considera-se que uma lista na qual **menos que 4** das respostas estão corretas não foi entregue. A entrega da lista com **pelo menos 7** das questões corretamente respondidas implica em um **acréscimo de 0,125 ponto** na média da disciplina para os membros da dupla. Se **qualquer situação de cópia de respostas for identificada**, os membros de todas as duplas envolvidas **perderão 0,5 ponto** na média da disciplina. O mesmo vale para respostas obtidas a partir da Internet. As respostas deverão ser entregues **exclusivamente em formato texto ASCII** (nada de .pdf, .doc, .docx ou .odt) e deverão ser enviadas para o monitor responsável por sua dupla, **sem** cópia para o professor. Devem ser organizadas em arquivos separados, um por questão, entregues em um único arquivo compactado, ou seja, um único arquivo .zip contendo as respostas para todas as questões. Um membro de cada dupla deve ir até a página da monitoria correspondente (CC ou EC) e registrar os nomes e logins dos membros da sua dupla sob o nome de um monitor. A escolha do monitor deve seguir uma política round-robin de modo a balancear a carga de duplas entre os monitores da maneira mais equitativa possível. A não-observância desta política pode resultar na transferência da dupla para outro monitor.

1) Merge sort é um famoso algoritmo de ordenação que utiliza estratégia dividir para conquistar. Implemente uma versão sequencial do Merge sort. Então, implemente uma versão que consiga tirar proveito do uso de threads.

Explicação Algoritmo:

1ª Fase: A lista inicial é dividida ao meio, consecutivamente, até chegar a valores unitários

2ª Fase: Os valores são comparados aos seus vizinhos, montando grupos maiores já ordenados.

Ex:
Lista: 3 5 7 4 9 10 2 1 12
1ª Fase
3 5 7 4 | 9 10 2 1 12
3 5 | 7 4 | 9 10 |2 1| 12
3| 5| 7| 4| 9|10| 2| 1| 12
2ªFase
3| 5| 7| 4| 9|10|2|1|12|
3 5 | 4 7 | 9 10 | 1 2 | 12
3 4 5 7 | 1 2 9 10 | 12

1 2 3 4 5 7 9 10| 12
1 2 3 4 5 7 9 10 12

(podem adotar outra estratégia para tratar valores não divisíveis por 2, mas devem colocar comentado)
Faça testes, compare e comente os resultados.

2) Uma thread recebeu a importante missão de transmitir uma mensagem formada por bits. Para isso ela dispõe de um repositório que pode abrigar no máximo 10 bits. Ela só consegue escrever um bit por vez. Adicionalmente 4 threads receptoras da mensagem devem acessar o repositório, mas elas também só conseguem ler um bit por vez. A mensagem deve ser enviada e recebida por inteiro na ordem exata e por todas as receptoras. As threads receptoras não devem impedir que outras receptoras façam seu trabalho ao mesmo tempo que elas. Faça um programa em haskell que implemente a thread responsável pelo envio, o repositório e as threads receptoras.

3) Elabore, usando MVars, uma simulador para o jogo Pilha de Três Cartas (http://bit.ly/1qP0RQV), jogado com três conchas e uma pedrinha, onde uma pedrinha fica sob uma das conchas que são embaralhadas, ao fim do qual o usuário é perguntado em qual das conchas está a pedrinha. O jogo porém é comandado por um polvo e este usa de seus oito tentáculos para mover as conchas dois a dois (por pares de tentáculos). Cada concha é representada por uma posição e um boleano indicando se possui ou não a pedrinha. Quando o polvo quer mover uma das conchas ele coloca um tentáculo sobre uma das conchas, então deve usar outro tentáculo (do mesmo par) para alcançar outra concha e trocá-las de posição. O programa deve continuar até que a pedrinha tenha mudado de posição n vezes (dado pelo usuário), então o usuário pode tentar adivinhar sua posição. Elabore sua solução de forma que os tentáculos não entrem em deadlock.

4) Implemente, usando MVars, um sistema de oferendas para uma pequena vila. A vila possui n fazendeiros e m deuses os quais devem receber as oferendas (m e n definidos pelo usuário). Cada fazendeiro produz oferendas e cada deus consome uma oferenda continuamente. Quando um deus vai consumir suas oferendas, porém não há nenhuma produzida, este mata um dos fazendeiros. Quando um fazendeiro vai produzir uma oferenda, mas a cesta de oferendas está cheia (ao atingir um limite x indicado pelo usuário), o fazendeiro volta pra casa, fica com sua esposa e produz um filho (este torna-se imediatamente um fazendeiro). O programa encerra quando os deuses tiverem consumido p oferendas (dado do usuário), ou quando não houver nenhum fazendeiro vivo. Em seu programa, deuses e fazendeiros devem ser modelados como threads.

5) Crie um programa em Haskell, utilizando TVars, que simule o problema do Barbeiro Dorminhoco (http://en.wikipedia.org/wiki/Sleeping_barber_problem). Em uma barbearia existe apenas um barbeiro, uma cadeira de barbeiro e n (defindo pelo usuário) cadeiras de espera. O barbeiro tem dois comportamentos: quando não há clientes esperando, ele cochila em sua cadeira de barbeiro; quando há clientes esperando, ele chama um cliente para

fazer sua barba, demorando 2000 milissegundos barbeando seu cliente. Quando um cliente chega (a cada 1000~3000 milissegundos, se o barbeiro estiver dormindo, ele o acorda e senta na cadeira de barbeiro; caso contrário, ele verifica se há vaga nas cadeiras de espera. Caso estejam lotadas, ele vai embora. Faça com que o programa acabe depois que m (definido pelo usuário) clientes tiverem a barba feita.

6) Lá está o Batman. Vários capangas do Coringa estão vindo em sua direção para lhe espancar. Faça um programa em Haskell utilizando MVars para simular essa briga. Cada capanga é uma thread com X(definido pelo usuário)HP e ataca Batman 1 segundo após conseguir se aproximar, tirando 1HP de Batman. Se o ataque ocorrer, estiver sendo atacado por Batman ou se já tiverem 3 capangas tentando atacar Batman, não é permitindo que o mesmo consiga se aproximar do morcego, daí ele espera entre 5 e 10 segundos para tentar novamente. No máximo 3 capangas podem atacar Batman ao mesmo tempo. Quando seu HP chegar a 0, o capanga cai e deixa de atacar. Tanto Batman quanto cada capanga são modelados como uma thread, com Y(definido pelo usuário)HP. Se nenhum capanga estiver atacando o morcegão, ele escolhe um capanga aleatório e começa a atacá-lo. Cada ataque tira 1 de HP do capanga, com cada ataque seguinte tirando 1HP a mais. Batman espera meio segundo entre ataque subsequentes. Antes de tentar atacar ou continuar um combo, se alguém estiver se preparando pra atacá-lo, ele contra-ataca todos os agressores ao mesmo tempo, tirando HP de todos igual à quantidade de agressores (ex: 1 agressor contra-atacado perde 1HP, 2 agressores perdem 2HP cada). Enquanto contra-ataca, Batman não pode ser atacado. Se Batman for atacar e só tiver 1 último inimigo, um ataque especial é desferido, nocauteando o capanga de vez independente do seu HP. O programa acaba quando todos os capangas perderem todo seu HP ou quando Batman perder todo o seu HP. Imprima uma mensagem sempre que Batman acertar um golpe ou for acertado por um capanga, mostrando o HP restante da vítima.

7) Na matemática, o problema "no-three-in-line" consiste em achar uma configuração de pontos em uma malha NxN, de modo que em qualquer linha (vertical, horizontal ou diagonal), haja no máximo 2 pontos. Construa um programa que, dado um número N, resolve de forma paralela o problema "no-three-in-line". Desenvolva também uma versão sequencial e compare os desempenhos quando variamos o valor de N e o número de threads empregadas. Seu programa deve produzir como resultado as coordenadas onde os pontos deverão ser colocados. Por exemplo, para uma malha 4×4, o programa pode produzir como saída:
[(1, 1), (1, 2), (2, 3), (2, 4), (3, 1), (3, 2), (4, 3), (4, 4)]
OBS: Para mais informações: http://en.wikipedia.org/wiki/No-three-in-line_problem

8) Para comer no R.U. da UFPE, além de enfrentar um sol de rachar, o aluno precisa lidar com a lentidão das atendentes. São três atendentes e uma fila única de atendimento. Se não há nenhum aluno na fila, os caixas esperam até que algum deles apareça. Se um aluno chega e vê que a fila está muito longa (definido por uma variável inicial que limita o tamanho da fila), ele vai embora comer no Dragão Expresso. Senão ele entra na fila e espera ser atendido. Seu

programa deve rodar até que um número máximo de clientes a ser atendidos (definido inicialmente) tenha sido atingido.

- Cada atendente deve ser implementada como uma Thread distinta. Quando uma atendente está livre e há alunos na fila, ela irá atender o primeiro da fila e o cliente entrará no R.U. (liberando um espaço na fila).
- Uma atendente pode levar de 25ms à 100ms para atender um aluno.
- A lista de clientes esperando deve ser mantida em um TVar.
- Alunos chegam em intervalos de 50ms a 500ms, definidos aleatoriamente.

Escreva informalmente e em forma de comentário as propriedades de segurança e vivacidade para este problema e mostre que seu programa satisfaz essas propriedades.