2ª Lista de Exercícios de Paradigmas de Linguagens Computacionais

Professores: Fernando Castor, Paulo Borba

Monitores: Marcos Paulo Barros Barreto (mpbb)
Tulio Paulo Lages da Silva (tpls)

CIn –UFPE - 2014.2
Disponível em: 09/01/2014
Entrega: 30/01/2014

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 6** das respostas estão corretas não foi entregue. A entrega da lista com **pelo menos 13** 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.

1) Faça um programa em java que dado um número N calcula se este número é primo. Um número primo tem apenas dois divisores: um e ele mesmo. O programa deve dividir o N pelos números de 2 a N, para descobrir a primalidade por força bruta. Faça também duas versões deste programa onde o calculo deverá ser realizado paralelamente por várias threads: uma onde a divisão de trabalho é feita dinamicamente e outra onde essa divisão é estátca. A realização do cálculo deverá ser dividida entre as threads criadas de modo que em um processador com X cores, o programa com X threads consiga ser mais rápido que o programa sequencial. O número de threads criadas será um parâmetro X do programa. Observe e descreva, em um arquivo texto, os tempos de execução do programa sequencial e para X igual a 2, 100, 500, 1000, 10000. A partir de qual valor de X a execução do programa ficou mais lenta que a sequencial? Explique o porquê.

2) Faça um programa em java no qual recebe duas matrizes quadráticas **nxn,** m1 e m2. Então roda m1 * m2 com 1 a **n** threads (Se aproveitando ao máximo do paralelismo) e retorna os tempos necessários para fazer os cálculos com os diferentes números de threads, ressaltando o que tiver gasto menos tempo.

3) 10 guerreiros mágicos compartilham uma fonte de poderes. Sua meta é reunir a energia necessária para juntos destruírem um mago das trevas, que ameaça invadir sua aldeia. Para

isso eles saem em pequenas caçadas individuais e retornam (ou não) trazendo energia para a fonte. Assim que a fonte atingir 1000 pontos energéticos, todos que ainda estiverem vivos se reunirão a partirão para tentar destruir o mago.

Cada guerreiro possui um parâmetro que determina sua força, variando entre 5 e 10. Esse valor representa seu ataque que nada mais é do que a porcentagem da fonte. (ex: se a fonto possui 100 de energia e um guerreiro 7 de força, seus ataques serão 7)

A fonte inicialmente possui 100 de energia. Deve existir uma lista de criaturas para os guerreiros caçarem onde cada criatura deve possuir. Valor minimo energético da fonte para que os guerreiros possam enfrentá-las, valor energético que ela oferece, sua vida e seu ataque.

As batalhas entre os guerreiros e criaturas acontecem por turnos, onde o ataque de um é removido da vida do outro até que algum deles chegue a 0. Sendo destruído. Se um gerreiro vence uma batalha ele volta para colocar a energia na fonte, favorecendo seus amigos. Então espera (0,1 segundos) para que sua vida volte ao máximo para então voltar a caçar.

Os guerreiros não tem tempo a perder e escolhem as criaturas que ofereçam o maior valor energético disponível para caçar.

A batalha dos guerreiros remanescentes contra o mago também segue a mesma ideia. Um turno todos os guerreiros desferem ataque contra o mago no outro o mago desfere o ataque.

O mago deve possuir. Sua vida inicial, capacidade de regeneração de vida por turno, ataque individual, ataque espalhado (Atinge todos os guerreiros). 30% das vezes o mago deve atacar espalhado. Faça um programa em java para representar a jornada dos guerreiros, onde cada um deles deve rodar em uma thread e os adversários em threads separadas. No final se deve imprimir se os guerreiros salvaram sua aldeia destruindo o mago, e quantos permaneceram vivos ou se o mago os venceu.

4) Analise a classe Deadlock abaixo e explique o que está causando o deadlock. Modifique o que achar necessário para encontrar uma solução para o problema de forma que a execução ainda seja feita com duas threads paralelas, fazendo uso de locks mas sem usar um lock global, sem fazer com que cada uma das threads subtraia ou some somente um dos números, sem modificar a lógica da classe. O resultado da execução deve tornar 'numPositvo' e 'numNegativo' iguais a 0. Modifique a questão para que ocorra o mesmo deadlock, mas faça uso de synchronized e não de locks.

Escreva num arquivo texto à parte a justificativa do que está causando o deadlock e um segundo arquivo java para a implementação do deadlock com o uso de synchronized.

```
import java.util.concurrent.locks.ReentrantLock;
public class Deadlock {
	ReentrantLock lock1 = new ReentrantLock();
	ReentrantLock lock2 = new ReentrantLock();
	long numPositivo = 3500000000l; long numNegativo = -3500000000l;
	public Deadlock() {
		Thread t1 = new Thread() {
			public void run(){ executar(lock1,lock2);}
		};
		Thread t2 = new Thread() {
```

```
                public void run(){ executar(lock2,lock1); }
            };
            try {
                t1.start();t2.start(); t1.join();t2.join();
            } catch (InterruptedException e) {}
        }
        public void executar(ReentrantLock fst, ReentrantLock snd){
            for (int i = 0; i < 50; i++) {
                fst.lock();
                try{
                    for (int w = 0; w < 35000000; w++) {
                        if(fst == lock1) numPositivo--;
                        else{ numNegativo++; }
                    }
                    snd.lock();
                    try{
                        for (int j = 0; j < 35000000; j++) {
                            if(fst == lock1) numNegativo++;
                            else{ numPositivo--; }
                        }
                    }finally{
                        snd.unlock();
                    }
                }finally{
                fst.unlock();
                try { Thread.sleep(200);}
                catch (InterruptedException e) {}
                }
            }
        }
        public static void main(String[] args) {
            Deadlock dead = new Deadlock();
            System.out.println("Número positivo se tornou : " + dead.numPositivo);
            System.out.println("Número negativo se tornou : " + dead.numNegativo);
        }
}
```

5) Utilizando os conceitos do problema do Produtor/Consumidor, desenvolva, usando locks explícitos, uma solução para o levantamento dos lanches comprados pelos clientes de uma lanchonete. A lanchonete tem três balcões onde são produzidos os lanches de forma independente. Os lanches podem ser diferentes, e por isso podem ter preços distintos. A quantidade de clientes (deve funcionar para pelo menos 100), o nome de cada cliente, a quantidade de lanches, o nome e preço de cada lanche são valores que devem ser definidos pelo usuário. Você pode considerar que há pelo menos três tipos de lanche

distintos (você os define) e que a escolha de qual será produzido em dado momento é feita aleatoriamente. Considere que os clientes são particularmente glutões e lancham com grande regularidade, a intervalos aletórios. Imprima na tela, para cada cliente, o nome, os lanches adquiridos e o total gasto. Considere que, em um dia de trabalho, a lanchonete serve pelo menos 10000 lanches.

6) Faça um programa em java que simule a evolução de um exercito de máquinas. Considere para isso uma máquina geradora capaz de criar soldados e operarios. A geradora precisa de 0,04 segundos e 150 kg de aço para criar um operário e 0,05 segundos e 200kg de aço para criar um soldado.

Cada operário é capaz de produzir 100kg de aço a cada 0,06 segundos, mas são danificados e perdem o funcionamento após 4 segundos, parando sua produção.

Devido a problemas estruturais o local da geradora suporta apenas 5 toneladas de aço e caso se atinja esse valor rodos os operários param de produzir aço. Fora isso a máquina geradora usa imãs muito potentes para pegar as barras de aço, dessa forma no momento em que ela faz a coleta nenhum operário pode estar despejando aço. Por outro lado o espaço é livre a vários operários podem despejar aço ao mesmo tempo.

Seu programa para após 30 segundos e imprime a quantidade de soldados produzida, a quantidade de operários desativados e operários ativos. O programa deve respeitar as propriedades de liveness e safety e adicionalmente nunca ficar impossibilitado de permanecer produzindo Ou seja não possuir aço suficiente para construir um operário e nenhum operário ativo para produzir aço.

O programa inicia com 1 tonelada de aço, nenhum soldado e um operário.

-Você fica livre para construir a estratégia de produção de soldados e operários, tentando maximizar a produção final de soldados. (não vai ser feita nenhuma comparação de produtividade para a nota da lista)

7) Quase todo aluno de Ciência da Computação tem sérios problemas em pronunciar Fila e Pilha em sequência. Pimenta, que é um ótimo aluno, resolveu acabar com esse problema criando uma nova estrutura de dados, a PilaFilha.

A PilaFilha consiste de uma lista encadeada que permite as operações padrão de Filas e Pilhas: queuePush(A junção de queue e push), dequeue, pop.

QueuePush: Coloca um elemento na cauda da lista.

Dequeue: Retira o elemento que se encontra na cabeça da lista.

Pop: Retira um elemento da cauda da lista.

Vendo que a implementação estava muito simples, Pimenta resolveu dificultar um pouco o seu trabalho, e decidiu que a classe PilaFilha deveria ser thread safe (Ele não pagou PLC, mas achou o termo bonito). Assim, sobrou pra você fazer essa implementação. Implemente duas versões da estrutura PilaFilha, uma utilizando apenas blocos sincronizados, e outra utilizando apenas travas explícitas.

OBS:

A estrutura deve permitir operações simultâneas sempre que possível. Descreva em forma de comentário as situações onde é impossível realizar operações simultâneas, e descreva brevemente o porquê das suas implementações serem realmente thread safe.

8) Em uma obra em estágio inicial os pedreiros contratados só receberam duas funções. A primeira é quebrar paredes das antigas instalações. A segundo desempenar algumas peças metálicas que serão usadas ao longo da construção. Para quebrar paredes os pedreiros precisam de um martelo e um ponteiro (ferramenta). Para desempenar as peças metálicas eles precisam ter acesso a um torno e um martelo. Faça um programa em java usando travas explicitas que simule o trabalho dos pedreiros. Cada pedreiro deverá rodar em uma thread e possuir uma lista de atividades. Seu programa deve receber o número de pedreiros, as listas de atividades de cada pedreiro, o número de martelos, ponteiros e tornos disponíveis. O programa termina quando todos os pedreiros tiverem executado todas suas atividades.

-Considere, para efeito da simulação, que um pedreiro demora 0,1 segundos para desempenhar uma peça e 0,2 segundos para quebrar uma parede.

-Seu programa deve possuir propriedades de Liveness e Safety

9) Uma empresa está desenvolvendo um jogo de briga entre gangues e você deve ajudar a montar a simulação entre as brigas. O jogo certamente não será permitido para menores, pois é muito violento. As poucas regras das brigas são. Em nenhum momento dois podem bater em 1 e dado que dois comecem a brigar só devem parar quando um deles estiver desmaiado. Ou seja se um membro de uma gangue ataca alguem, esse segundo passa a revidar e os dois entram em um combate que só acaba com um deles desmaiado. Cada membro de gangue deve rodar em uma thread e buscar aleatoriamente por adversários livres para brigar. Para impedir que dois comecem a brigar com o mesmo adversário você deve usar unicamente compareAndSet. A briga termina quando todos os adversários de uma gangue estiverem desmaiados. Considere que cada membro de gangue começa com 100 de vida e desmaia com 10 ou menos. Cada ataque desferido varia entre tirar 3,4,5 ou 6 de vida do adversário, caso tire 6 repete o ataque, caso contrário o adversário atacará. Após vencer um adversário o membro de gangue está pronto para outro combate e volta a busca por um adversário livre, mas sua vida não é regenerada.

-Deve-se imprimir no console toda vez que um membro de gangue desmaiar, explicitando a gangue que ele pertence (defina uma nomeclaruta, se preferir simplificar pode ser primeira e segunda)

-No final deve imprimir a gangue vencedora

- O número de membros de cada gangue é um parâmetro do programa e pode diferir um do outro.

10) 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.

11) 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.

12) Na cidade de Raccoon, um vírus (chamado T-vírus) se espalhou pela cidade contaminando quase toda a população. O T-vírus depois de certo tempo transforma as pessoas infectadas em zumbis. A única saída para uma pessoa infectada é ir para a base militar da cidade, onde o governo montou N (definido pelo usuário) salas de descontaminação. Entretanto, é preciso enfrentar uma fila de prioridades na qual as pessoas mais influentes têm a preferência1.
Considere as seguintes informações:
- Cada Infectado dever ser implementado como uma thread distinta e terá as seguintes características.
- Um inteiro representando a ordem de chegada do infectado à base (ID).
- Um inteiro (de 1 a 5) representando sua influência (deve ser definido aleatoriamente nas seguintes probabilidades): 5-Magnata (5%), 4-Polítco (10%), 3-Celebridade (20%), 2- Militar (25%), 1-Cidadão Comum (40%). Ex: se aleatoriamente fosse gerado um numero entre 0 e 1, e este numero ficasse entre 0 (inclusive) e 0,4 (exclusive) o infectado seria um cidadão comum, se ficasse entre 0,4(inclusive) e 0,65 (exclusive) o infectado seria um militar e assim por diante.
- Um inteiro (> 0 e < 100, deve ser defnido aleatoriamente) representando a grau de infecção do infectado ao chegar à base. - Enquanto o tempo passa o infectado piora (incrementando seu grau de infeção em 10 a cada 1000ms) até chegar a um estado irreversível no qual ele vira um zumbi (Ocorre quando o grau de infecção fica maior ou igual a 100). - A fila de espera será representada por um Heap, com capacidade X de infectados (definido pelo usuário). É obrigatória a implementação do seu próprio Heap e que ele seja seguro para threads. Utilize MVars.

13) Você está programando um servidor que cria partidas de dominó online. Em cada partida, apenas 4 pessoas jogam. Implemente o servidor de tal forma que as partidas são criadas a cada 4 requests de jogadores. Um novo jogador tenta jogar a cada 60ms ou 120 ms e o servidor consegue criar apenas 2 partidas por segundo. Se um jogador tentar entrar e a fila de espera do servidor estiver cheia (definida pelo usuário), ele desiste e vai jogar paciência. Imprima no console toda vez que uma pessoa tentar jogar e todo jogo que o servidor formar. Além disso, relate no final da execução do programa a quantidade de jogos formados e a

quantidade de usuários que não conseguiram jogar! Seu programa deve rodar por um tempo fixo, definido pelo usuário. Utilize TVars na solução do problema.
Exemplo de saída do programa (fim):
User: Happy guy gonna play dominoes!
(...)
User: Happy guy gonna play dominoes!
(...)
User: Sad dude gonna play solitaire!
(...)
User: Sad dude gonna play solitaire!
(...)
Server: match created
(...)
User: Happy guy gonna play dominoes!
(...)
User: Happy guy gonna play dominoes!
(...)
20 matches were created!
But 28 went to play solitaire =(
Acima, o símbolo “(...)” indica que pode haver outras linhas entre a anterior e a posterior.

14) Seu Zé possui uma fábrica de chocolates muito boa, só que infelizmente, a demanda da fábrica é muito grande e ele não consegue supri-la. Sabendo disso, Seu Zé, muito ganancioso, decide automatizar o processo com máquinas e estuda a compra de dois tipos de máquinas. O primeiro tipo máquina é uma chocolateira. Ela recebe os ingredientes brutos e fabrica o chocolate, entregando 100 chocolates a cada vez que consegue acesso à esteira. O segundo tipo trata-se de uma embaladeira, que recebe chocolates prontos e coloca-os na embalagem, deixando-os prontos para a venda. A embaladeira consegue embalar 80 chocolates a cada vez que consegue acesso a esteira. A esteira da fábrica possui espaço para apenas 400 chocolates. Como as máquinas são muito caras, Seu Zé convenceu o professor de PLC do CIn a passar um simulador da fábrica como uma questão da lista 4 da disciplina. Esse simulador deve receber como entrada a quantidade de máquinas do tipo 1 e do tipo 2 e deverá rodar até que sejam fabricados 100000 chocolates. Deve gerar um log ao final da execução da quantidade de vezes que a esteira ficou cheia ou vazia, para que Seu Zé tenha uma ideia de quão improdutiva está sendo a configuração testada. Seu programa deve usar TVars para garantir a concorrência das máquinas.

15) Um carro de fórmula um, ao parar no Pit Stop, troca seus pneus e coloca combustível. Existem cinco funcionários que realizam essa tarefa, cada um representado por uma Thread. Cada funcionário é altamente especializado e sabe apenas realizar uma das seguintes operações:
1 - Analisar se um pneu precisa ser trocado (20 milissegundos, com a probabilidade de 50%), e removê-lo (20 milissegundos)

2 - Colocar um novo pneu (50 milissegundos)
3 - Colocar combustível (200 milissegundos)
Existem dois funcionários responsáveis pela tarefa 1, dois pela tarefa 2, e um pelatarefa três.
O combustível não pode ser colocado enquanto houver algum pneu sendo removido (o funcionário que estiver colocando o combustível interromperá a operação e a retomará quando for possível) e dois funcionários não devem trabalhar numa mesma roda.
Faça uma simulação de um carro passando 5x nesse Pit Stop, imprimindo o que cada funcionário fez e o tempo que o carro levou em cada parada. Use TVars.