**Universidade Federal do ABC**

**Exercício programático 1**

**EP1**

**Aluno: Ramon Neres Teixeira Jardim RA: 11068616**

**Professor: Vladimir Emiliano Moreira Rocha**

**Santo André**

**2019**

**Formato da mensagem transferida entre *peers* via *gossip*.**

A carga de dados transferida entre os clientes é um objeto java serializado em um vetor de bytes. A classe deste objeto foi nomeada "PeerInfo" e possui 5 propriedades:

* ***PeerName***: Uma *String* contendo o nome do *peer* ao qual este objeto representa;
* ***FilesInfo***: Uma lista de Metadata, contendo as informações dos arquivos;
* ***InfoNumber***: Número inteiro identificando a versão da mensagem, permitindo diferenciar mensagens novas, antigas e duplicadas.
* ***ReceiveMoment***: Um *long* armazenando o momento em que a mensagem foi recebida, permitindo identificar mensagens antigas e apagá-las.
* ***PeerSender***: Uma *String* contendo o nome do *peer* que enviou a mensagem.

**Explicação em “alto nível” do *gossip* implementado.**

Os *peers* possuem 6 threads que utilizam um *singleton* chamado DAO para armazenar as informações em memória. As *threads* são:

* ***Receiver***: “Escuta” por pacotes UDP na porta 9876. Ao receber um pacote, o conteúdo é convertido para um *PeerInfo* e armazenado em um dicionário no DAO, utilizando como chave o nome do *peer* ao qual as informações se referem;
* ***FileModifier***: Modifica aleatoriamente a pasta de arquivos.
* ***SelfStateReader***: Acessa o sistema de arquivos a cada 500 ms e guarda as informações em um objeto *PeerInfo* no DAO;
* ***SelfStateSender***: Envia periodicamente as informações obtidas pelo *SelfStateReader* para um *peer* aleatório;
* ***ForeignStateSender***: Envia periodicamente e de forma aleatória as informações recebidas pelo *Receiver* para um *peer* aleatório. O *peer* alvo é diferente daquele que gerou as informações e do que as enviou;
* ***OldStateCleaner***: Constantemente compara o campo *receiveMoment* das informações armazenadas no DAO com o tempo atual do sistema, removendo a informação do dicionário caso ela tenha sido recebida a mais de 1500 ms.

**Explicação em “alto nível” do tratamento de mensagens duplicadas ou antigas.**

Todos os pacotes recebidos na *thread Receiver* são enviados para o DAO paraque suas informações sejam inseridas no dicionário. No DAO, são feitas duas verificações caso a chave (nome do *peer*) já exista no dicionário. Se o valor do campo *infoNumber* da mensagem recebida for igual ao valor deste campo na mensagem já armazenada, o DAO lança uma *RepeatedInfoException*, e caso o *infoNumber* seja menor do que o já armazenado, é lançada uma *OutdatedInfoException*. Ambas as exceções são tratadas no *Receiver*, que informa no console o ocorrido ou sucesso quando nenhuma exceção é lançada.

**Links dos lugares de onde baseou seu código (caso aplicável).** O uso do ConcurrentHashMap foi baseado em: <https://crunchify.com/hashmap-vs-concurrenthashmap-vs-synchronizedmap-how-a-hashmap-can-be-synchronized-in-java/>

Os métodos para serialização e desserialização de um objeto para uma *array* de *bytes* foi retirado de: <https://stackoverflow.com/questions/2836646/java-serializable-object-to-byte-array>.

O código para escrever texto em arquivos foi baseado em: <https://www.javatpoint.com/javafilewriter-class>.

Os códigos para envio e recebimento de pacotes foram baseados nos slides de aula e em implementações feitas na disciplina de Redes de Computadores.