Sistemas Distribuídos

Relatório do Primeiro Projeto – Serviço de Backup Distribuído

Grupo T6g08:

Mariana Duarte Guimarães - up201307777

Tiago André Pérola Filipe – up201610655

Introdução:

Neste relatório será explicado como foi desenhada e implementada a concorrência no nosso projeto, ou seja, o código que permite executar instâncias dos protocolos de forma concorrente.

Concorrência:

Relativamente à concorrência na execução de tarefas do projeto, sendo que uma das partes principais é um peer estar à escuta nos três canais de multicast, optou-se por fazer esses "listenners" em threads diferentes, com o objetivo de quando o peer estiver à escuta de mensagens nunca ficar impedido de executar outras tarefas.

Como existem várias tarefas que necessitam de ser executadas quando alguma mensagem é recebida num canal multicast, houve a necessidade de lançar um EventHandler, quando o canal recebe um pacote, inicia uma thread que trata desse pacote recebido.

Para além de estar à escuta das mensagens que vários peers enviam através desses canais, o programa também consegue em simultâneo estar a aguardar por pedidos do cliente. Sempre que um pedido de protocolo é recebido, esse pedido é resolvido numa nova thread à parte.

Posto isto, não chega apenas uma boa estrutura de threads para o programa ser eficiente, existe também a necessidade de utilizar ferramentas que garantam a execução com sucesso das tarefas das threads. Para isso, no que toca a acesso de memória run-time partilhada entre threads utilizaram-se as seguintes classes da biblioteca concurrent do Java: ConcurrentHashMap e CopyOnWriteArrayList que garantem um acesso thread-safe aos dados. Sobre a escrita da memória não volátil utilizaram-se os métodos syncronized do Java. A solução ideal seria utilizar a classe AsynchronousFileChannel, mas surgiram algumas dificuldades e para o desenvolvimento do trabalho não atrasar optou-se por abdicar desta solução.

Para finalizar, tal como recomendado, evitou-se a utilização do método sleep pertencente à classe Thread e utilizou-se a classe ScheduledThreadPoolExecutor pertencente à biblioteca concurrent para se efetuar os timeouts das threads. Quando era necessário executar uma tarefa após um determinado intervalo de tempo, agendava-se essa tarefa para iniciar quando esse intervalo de tempo termina-se.