Relatório do primeiro projeto de Sistemas Distribuidos

Participantes

Este projeto foi realizado pelos seguintes alunos:

- Pedro Azevedo (up201806728)
- João Cardoso (up201806531)

Concurrency design

Our program utilizes the concurrency strategies described in points 4, 5 and 5.1 of your concurrency strategy listing.
This means that, firstly, each multicast thread has a different thread assigned to itself, meaning that all three (MC - Multicast Control, MDB - Multicast Data Backup and MDR - Multicast Data Recovery) channels can receive messages at the same time.
In addition, we utilize a thread for each message received, different from the multicast threads, allowing us to handle multiple requests at the same time.
In order to avoid the repeated creation and destruction of these last threads, we use the services provided by the class java.util.concurrent.ThreadPoolExecutor, which manages a set number of threads to handle these messages.

Implementation of said design

Each of the program's peers contains 3 instances of objects from the class MulticastThread, each with a string that identifies which of the channels it is (MC, MDB or MDR)
Whenever any of the channels receive a message, they immediately send the message to one of the threads from the ScheduledThreadPoolExecutor[1].
Each of those threads runs an instance of the "void run()" method of the "MulticastHandler/MulticastResponseHandler" class, in which each message is interpreted and managed (including handling of "metafiles"[2] and chunks), if one is needed, a response message is created and sent through the appropriate multicast channel.

[1] - We send the message to the thread without checking it's contents because the report specifies this product will not have malicious users, which made us decide that the channel's message receiving speed is more important than the time wasted by running the threads when a message should be ignored.
[2] - Files that contain information on a peer's backed up files and stored chunks