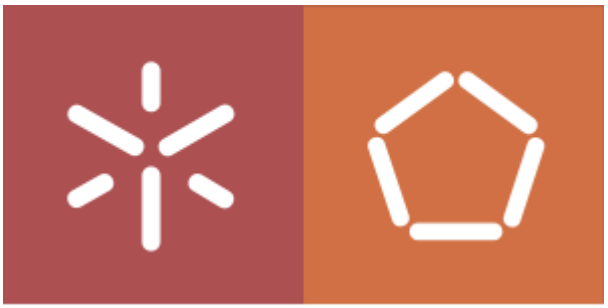

Universidade do Minho
Escola de Engenharia

Licenciatura em Engenharia Informática

Sistemas Distribuídos

Grupo 20
a97903, José Fernando Monteiro Martins
a100535, Gonçalo Nuno da Silva Loureiro
a100599, Gonçalo Antunes Corais
a100604, Afonso Oliveira e Silva

Trabalho prático: Cloud Computing

Dezembro 2023

Índice

Introdução

No âmbito deste projeto, foi-nos proposta a criação e implementação de um serviço de cloud computing, centrado na funcionalidade de Function-as-a-Service (FaaS). Este teve como objetivo principal permitir que um cliente consiga enviar tarefas para serem executadas em servidores remotos, usufruindo da memória disponível.

O projeto foi dividido em três partes, sendo estas a sua funcionalidade básica, funcionalidade avançada e a implementação distribuída. No nosso caso, todas elas estão implementadas.

A arquitetura do sistema é composta por um cliente local que comunica com um servidor principal que estabelece e mantém a conexão com o cliente e que utiliza outros servidores cujo único propósito é executar as tarefas que lhe são enviadas e devolver o resultado das mesmas. A comunicação é segura e com controlo próprio de erros, fluxo e congestão, utilizando o protocolo *TCP/IP*. Este sistema funciona da seguinte forma: o cliente comunica com um servidor principal que distribui as tarefas que recebe dos clientes pelos servidores disponíveis, mantendo em cada um uma fila de espera de tarefas a ser executadas, desde que as mesmas não ultrapassem a capacidade dos servidores, garantindo assim um uso eficiente dos recursos disponíveis.

De forma a haver comunicação entre o servidor e o cliente decidimos utilizar sockets, que são interfaces de programação de aplicativos que permitem que os programas comuniquem entre si, recorrendo a threads ,por exemplo, no servidor principal para conseguirmos atender vários clientes ao mesmo tempo. As threads são unidades de execução em um programa, totalmente independentes umas das outras e que permitem a execução das instruções de forma paralela, aumentando a rapidez do nosso serviço.

Menu

Para que um utilizador consiga interagir com o serviço, é necessário que este tenha uma conta associada a si e inserida na nossa base de dados, caso contrário vai lhe ser pedido que crie uma. Em caso de inserção incorreta de dados, um erro irá aparecer para cada caso específico que tenha provocado esse mesmo erro como, por exemplo, *password* incorreta, *username* que não se encontre na base de dados ao fazer login, entre outros.

Uma vez autenticado, um utilizador tem as opções (Figura 2):

- Correr uma tarefa (Figura 3);
- Ver os resultados das suas tarefas (Figura 4);
- Verificar o estado de ocupação do sistema (Figura 5).

Cliente

O Cliente é uma classe capaz de comunicar com o servidor principal através de conexões TCP. Por esse motivo, é através das funções loginUser() e registerUser() que vai ser enviada uma solicitação de login e registo, respetivamente, ao servidor, retornando um código de resposta positiva ou negativa. Relativamente às outras funções, a todoFiles() retorna o número de tarefas pendentes no servidor, freeSpace() retorna a quantidade de

memória disponível e a sendCode() envia as informações sobre a tarefa a ser executada e retorna um código de resposta e o tempo esperado de execução.

ClientFileInfo

A classe ClientFileInfo tem um papel importante na comunicação cliente servidor, nomeadamente na função sendCode(). Esta classe guarda informações sobre o cliente que solicitou a execução de uma tarefa, o código a ser executado, a hora a que a solicitação foi feita e, opcionalmente, o nome que o cliente deseja que o ficheiro com o resultado da execução tenha. Para além disto esta classe fornece também métodos como a verificação se o arquivo associado existe na máquina local (fileExists()), lê o conteúdo do mesmo como um array de bytes (getCode()) e calcula o tempo esperado necessário para a execução do código (getEstimatedTime()), em circunstâncias favoráveis em termos de recursos do servidor e calcula ainda o uso esperado de capacidade do servidor em bytes (getMemoryUsage()).

ClientHandler

A classe ´ClientHandler´ é fundamental para o funcionamento do nosso sistema, pois fornece a capacidade de lidar com múltiplos clientes, cada cliente que estabelece conexão com o *socket* do servidor principal é tratado por meio de uma instância da classe ´ClientHandler´ que estende a classe ´Thread´. O código abrange funcionalidades de autenticação, manipulação de arquivos e execução de tarefas.

MainServer

A classe ´MainServer´ inicia o nosso servidor principal, que cria instâncias para controlar os arquivos concluídos, arquivos por concluir, servidores auxiliares para executar as tarefas (com diferentes threads para cada um) e clientes. Para garantirmos o bom funcionamento e a sincronização entre as threads usamos as classes ´ReentrantLock´ e ´ReentrantReadWriteLock´, que garante que os objetos que são partilhados pelas várias threads são acedidos apenas por uma de cada vez (ReentrantLock) ou que permitem que várias threads tenham acesso simultâneo de leitura às variáveis da classe, mas apenas uma thread pode ter acesso para alterar os valores das variáveis, não podendo nenhuma outra ter acesso às variáveis enquanto uma estiver a alterar os seus valores (ReentrantReadWriteLock).

Resumindo esta classe, o seu código cria um sistema de processamento distribuído onde clientes enviam tarefas, o servidor principal coordena a execução entre os seus servidores auxiliares, e o compartilhamento das classes garante a consistência dos dados num ambiente competitivo e concorrente.

OutputFileInfo

Esta classe representa informações relacionadas à saída de uma tarefa executada. É bastante semelhante à classe ClientFileInfo, mas direcionada ao *output* da tarefa e não ao código da tarefa a ser executada.

ServerSlave

Esta classe representa a implementação de um servidor secundário, cujo único propósito é executar as tarefas que recebe e devolver o resultado das mesmas. Estes servidores comunicam apenas com o servidor principal e toda a sua comunicação é num formato binário recorrendo apenas a Data[Input|Output]Stream. Esta classe implementa o controlo de erros na execução de uma tarefa e a comunicação da ocorrência dos mesmos. Implementa ainda um controlo do tempo de execução de cada tarefa, onde após um certo período de tempo o servidor determina que o código está a executar há demasiado tempo e para a sua execução, de forma a poupar os seus recursos e a combater erros no código como, por exemplo, ciclos infinitos. Esta classe implementa também ReentrantReadWriteLocks, para que várias threads consigam ver se o ServerSlave está disponível para executar uma tarefa (aceder a uma das suas variáveis) mas permite também que uma thread possa adquirir o lock para escrita e que nenhuma outra thread aceda às variáveis enquanto ela não libertar o lock.

ServerSlaves, ToDoFiles, DoneFiles

Estas classes representam objetos do nosso sistema que são partilhados por todas as threads, cada uma delas com apenas uma instância criada no MainServer . Como tal, cada classe contém um ReentrantLock para garantir que apenas uma thread acede de cada vez às variáveis da classe, de forma a garantir que não existem *data races,* permitindo dessa forma que o nosso código seja fiável, a todo o momento de leitura e de escrita.

Clients

Esta classe, embora similar às classes acima descritas, está separada pois esta utiliza ReentrantReadWriteLocks, para permitir que várias threads tenham simultaneamente acesso de leitura às variáveis da classe, mas quando uma thread quiser alterar as variáveis pode adquirir o lock para escrita e apenas essa tem acesso nesse momento às variáveis da classe. Por exemplo, nesta classe dá-nos jeito para que várias threads possam fazer pesquisas no HashMap de clientes ao mesmo tempo e que apenas uma tenha acesso quando é preciso alterar o mesmo.

StressTest

Esta classe foi implementada para testar o nosso sistema com uma quantidade de pedidos com um pequeno intervalo de tempo de separação, que manualmente não conseguimos criar, de forma a verificarmos se o sistema aguentaria vários pedidos ao

mesmo tempo, se os devolveria com o nome correto e se os locks estariam bem implementados, não permitindo às threads acederem ao mesmo tempo a objetos partilhados.

Menu Interativo

O Menu Interativo permitirá que se faça uma escolha e visualização das funcionalidades implementadas no nosso projeto. Quando iniciado, irá pedir ao utilizador que faça o login ou que, caso ainda não tenha conta, se registre.

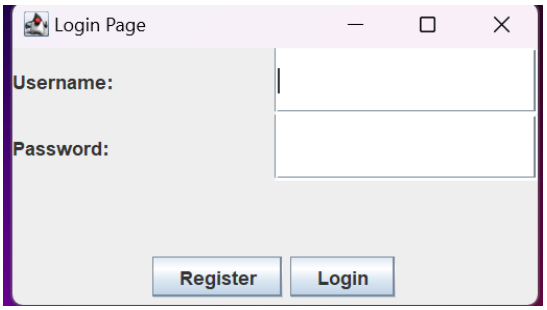

Figura 1 - Menu de autenticação do sistema.

Após feito o login, o utilizador irá poder escolher as opções apresentadas na seguinte imagem:

Figura 2 - Menu principal do sistema.

No caso de escolher a opção *Run Code,* o sistema pede ao utilizador que indique a direção para os ficheiros pretendidos, assim como os nomes para os mesmos (retângulo à esquerda na Figura 3) ou que os insira diretamente a partir do seu computador (retângulo ao meio na Figura 3).

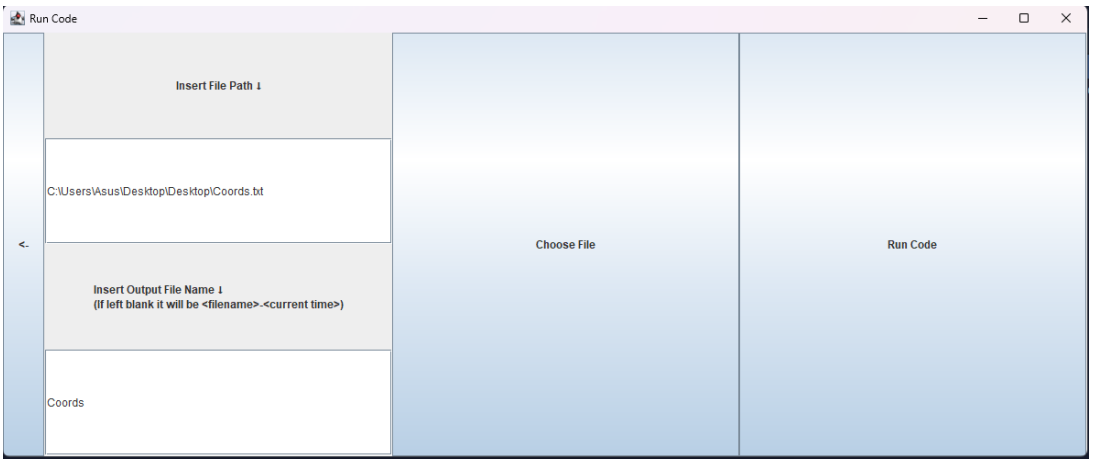

Figura 3 - Opções de inserção de ficheiros para serem corridos.

Após correr o programa, os outputs, resultados, tarefas em espera ou a memória disponível, serão devolvidos sendo que os mais recentes aparecem primeiro.

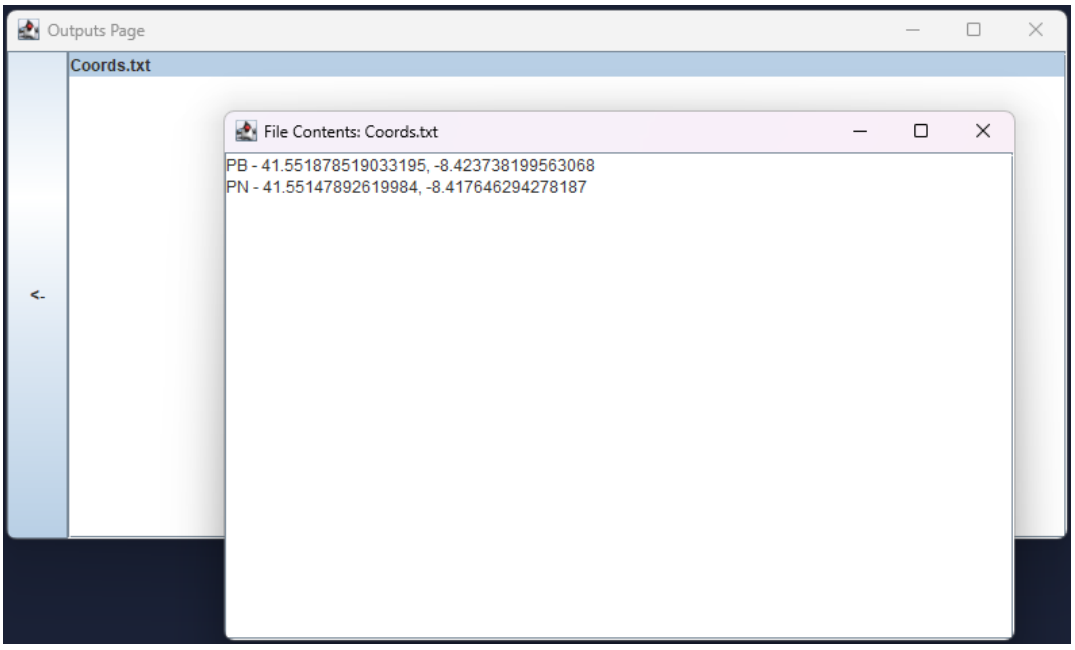

Figura 4 - Resultados das tarefas.

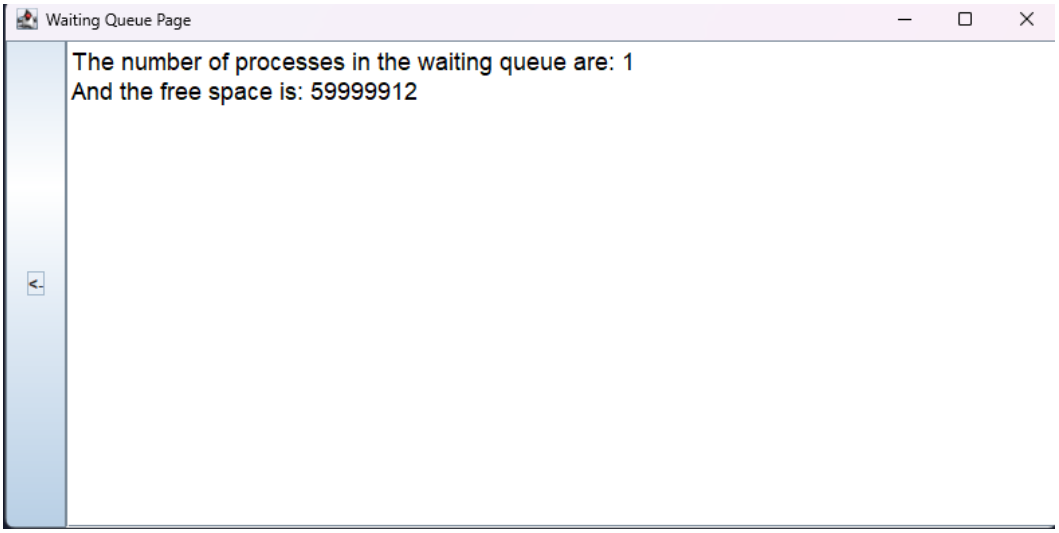

Figura 5 - Número de tarefas em espera e memória disponível.

Conclusão

Este trabalho permitiu-nos consolidar a matéria lecionada nas aulas de Sistemas Distribuídos como a comunicação TCP/IP entre cliente e servidor através de *sockets* e a execução de várias tarefas em simultâneo recorrendo às *threads,* e assegurando a não existência de *data races* utilizando vários tipos de locks.

Relativamente ao projeto em si, conseguimos implementar todas as funcionalidades pedidas e, por esse motivo, acreditamos que foi bem conseguido. Tivemos alguns obstáculos na implementação do trabalho, principalmente na comunicação entre o servidor principal e os servidores secundários (funcionalidades avançadas) e ao garantir a granularidade de locks, mas na sua grande maioria a realização deste projeto correu bem e, nos nossos testes, o sistema está com todas as funcionalidades funcionais e a respeitar os princípios fundamentais lecionados na cadeira.