Relatório Rede de Computadore

Erick Parras - l50059

18 de junho de 2023

1 Introdução

O trabalho tinha como objetivo criar um servidor capaz de fazer transferência de arquivos, broadcast entre utilizadores, possuir um sistema de perguntas e respostas e uma base de dados para guardar a presença dos participantes, e pela rara oportunidade de poder escolher a linguagem de programação a utilizar, decidi aproveitar a chance para aprofundar meus conhecimentos de Python e possuir todas as ferramentas e bibliotecas necessárias sem precisar adicionar módulos externos (para o nível do que o trabalho pedia).
Foi realizado com sucesso todas as quatro partes principais do trabalho e uma das opcionais.Foram utilizadas 8 bibliotecas preexistentes no Python, além de duas classes criadas para os serviços específicos, entre diversas funções que são explicadas no decorrer do trabalho.

2 Gestão de presenças

O sistema de presenças foi realizado usando o "sqlite3", um sistema de base de dados que já vem disponível no Python desde sua versão 3.0, sendo um sistema eficaz de query e armazenamento de dados. Foi utilizada a seguinte tabela para as presenças:
"CREATE TABLE IF NOT EXISTS clientes (
login text PRIMARY KEY,
presenca integer,
atrasado integer,
falta integer)"

O sistema base do servidor, tanto quanto do cliente, foi montado com base no que foi estudado em sala de aula, usando sockets TCP e multithreading. Porém, uma diferença é que o cliente possui duas threads separadas, uma dedicada a ler as mensagens do servidor (por meio da função receive()) e outra dedicada a enviar mensagens (a função send()), dessa forma cada cliente pode interagir livremente podendo receber mais de uma mensagem por vez por exemplo.

2.1 IAM e o sistema de presença

Como indicado no enunciado, quando o servidor é iniciado, começa a contar o tempo para o início da aula. Isso é feito usando os módulos "datetime"e "timedelta", que permitem realizar cálculos e comparações entre o horário de início da aula e o momento em que um usuário marca presença usando o comando "IAM apelido". O apelido é armazenado para uso posterior. Adicionalmente toda vez que alguem se identifica, seu nome e buscado na base de dados, se não for encontrado alguem com o mesmo apelido, e criado uma entrada para e ja comeca a anotar sua presenca.

Os clientes que chegam com menos de 20 minutos de atraso recebem um valor de presença. Aqueles que chegam entre 20 e 45 minutos de atraso recebem um atraso (equivalente a meia-presença). Após 45 minutos, é considerada uma falta. No entanto, mesmo com atraso ou falta, os clientes ainda podem interagir normalmente com o servidor.

3 Gestão de perguntas e respostas

Para o gerenciamento das perguntas e respostas foi utilizado OOP, criando uma classe chamada "Questions.py". Ela é criada atribuindo um número à pergunta e, caso receba respostas, guarda o apelido e a resposta de cada pessoa em uma lista interna. Para o gerenciamento de todas as dúvidas, foi criada uma lista de Questions.

3.1 ASK

Quando uma dúvida é feita, é preparada uma mensagem para broadcast, informando a todos que a dúvida foi realizada e fornecendo seu número, caso alguém deseje responder. Em seguida, a dúvida é adicionada à lista de dúvidas junto com as demais, e a lista é salva em um arquivo externo (utilizando o módulo "pickle") como backup.

3.2 ANSWER

Similarmente, quando uma pergunta é feita, a string é preparada para ser usada em uma mensagem e, em seguida, é adicionada à lista de perguntas juntamente com o apelido da pessoa que a submeteu. Essa preparação é feita para o próximo método a ser realizado. Após ser adicionada à lista de perguntas, o sistema salva novamente todas as perguntas e respostas.

3.3 LISQUESTIONS

A LISQUESTIONS, por mais simples que seja, foi necessário criar uma classe (que acabou sendo reutilizada no futuro) para poder montar a mensagem que seria enviada aos usuários corretamente. Essa mensagem tem sido a razão para as perguntas serem preparadas quando são salvas na lista. Dessa forma, o LISQUESTIONS se resume a ciclos que vão concatenando uma na outra, indo a cada dúvida e para cada dúvida listando suas respostas e quem as enviou de forma eficiente, e no final fazendo um broadcast para todos os clientes.

4 Gestao de ficheiros

Para a gestão de arquivos, foi necessário usar a biblioteca "os", que dá acesso aos arquivos externos do computador. Foi reutilizada a classe de suporte criada para a "LISTQUESTIONS", chamada "MessageBuilder".

4.1 PUTFILE

Para o "PUTFILE", primeiro é preparado o arquivo que será escrito, e em seguida é aberto um while loop que continuará escrevendo no arquivo todas as informações que forem enviadas até atingir o limite de bytes indicado pelo tamanho do arquivo.

4.2 LISTFILES

O comando mais simples entre os tres de gestao de ficheiros, e assim como o "LISTQUESTIONS"e necessario utilizar a classe extra para ir contactenando o nome de cada ficheiro, uma vez que e utilizado ciclos "for"e o scope nao permite ser feito de outra forma.

4.3 GETFILE

O GETFILE ocorre quando um cliente seleciona um arquivo para baixar. O usuário seleciona o arquivo com base nos números/ordem apresentados no LISTFILES, e um contador é usado até chegar ao arquivo correto. Ao chegar ao arquivo correto, utiliza-se a classe "MessageBuilder"para passar uma string de um escopo para o próximo, montando a mensagem de aviso de quando o arquivo é enviado. Para enviar o arquivo após selecioná-lo, é usado um loop com seu tamanho máximo, e ele é enviado em partes (se necessário) até chegar ao fim.

5 Persistência de dados

Havia duas possibilidades para realizar a persistência de dados no servidor Python. Uma delas seria utilizando o módulo "schedule", que precisa ser baixado separadamente, e determinar um intervalo de tempo para executar uma função que poderia salvar o progresso. A outra forma adotada foi a seguinte: a cada vez que ocorre uma mudança na lista de dúvidas, utiliza-se o módulo "pickle", que permite salvar objetos em arquivos externos. Além disso, foi alcançado o objetivo opcional de guardar as marcações de presença. Isso é feito criando um banco de dados no "sqlite3"e conectando-o ao se conectar ao servidor.

6 Extra e conclusão

Adicionalmente, foi adicionado um comando adicional, "EXIT", que permite que um cliente saia facilmente do servidor, garantindo que não ocorram erros.

Em conclusão, o trabalho foi realizado com sucesso, alcançando todos os objetivos principais e também realizando um objetivo adicional opcional.