Laboratório 2 de Sistemas Distribuídos - 2020.2 - UFRJ
Aluno: Rodrigo Carvalho de Figueiredo
DRE: 117053497

Atividade 1

Objetivo: Refinar a arquitetura de software — usando o estilo arquitetural em camadas — apresentada abaixo.

Camadas:

1. Funcionalidades da camada de interface com o usuário : recebe do usuário o nome do arquivo de busca e exibe na tela o resultado do processamento. O resultado do processamento poderá ser: (i) uma mensagem de erro indicando que o arquivo não foi encontrado; ou (ii) a lista de palavras com suas ocorrências.

A lista de palavras virá da camada de processamento pronta para exibição na camada de interface com o usuário, visando concentrar todo o processamento e a formatação do resultado na camada de processamento, poupando assim a camada de interface de qualquer tarefa mais demandante.

2. Funcionalidades da camada de processamento: solicita o acesso ao arquivo texto. Se o arquivo for válido, realiza a contagem das palavras e prepara a resposta para ser devolvida para a camada de interface. Se o arquivo for invalido, responde com a mensagem de erro.

A lista de palavras será entregue como uma string representando o array das 10 palavras mais frequentes, em ordem decrescente de frequência, da seguinte forma:

“**[(‘palavra-1’, 9), (‘palavra-2’, 7), (‘palavra-3’, 6), …, (‘palavra-10’, 1)]**”

onde em cada par (chave, valor) a chave é uma string que representa a palavra em questão e o valor um inteiro que representa a quantidade de ocorrências dessa palavra no arquivo de texto lido.

Essa string será encapsulada em uma mensagem que será enviada para a camada de interface com o usuário por meio de comunicação via socket.

3. Funcionalidades da camada de acesso aos dados: verifica se o arquivo existe em sua base. Se sim, devolve o seu conteúdo inteiro. Caso contrário, envia uma mensagem de erro.

Atividade 2

Objetivo: Refinar a proposta de instanciação da arquitetura de software da aplicação definida na Atividade 1 para uma arquitetura de sistema cliente/servidor de dois níveis, com um servidor e um cliente, apresentada abaixo.

Proposta de arquitetura de sistema:

1. Lado cliente: implementa a camada de interface com o usuário. O usuário poderá solicitar o processamento de um ou mais arquivos em uma única execução da aplicação: o programa espera pelo nome do arquivo, faz o processamento, retorna o resultado, e então aguarda um novo pedido de arquivo ou o comando de finalização.

2. Lado servidor: implementa a camada de processamento e a camada de acesso aos dados. Implemente um servidor iterativo, isto é, que trata as requisições de um cliente de cada vez, em um único fluxo de execução (estudaremos essa classificação depois). Terminada a interação com um cliente, ele poderá voltar a esperar por nova conexão. Dessa forma, o programa do servidor fica em loop infinito (depois veremos como lidar com isso).

1) A estrutura de dados usada no servidor para armazenar as palavras mais frequentes em ordem decrescente de frequência será um dicionário da linguagem Python do tipo Counter (documentação).

As mensagens trocadas entre cliente e servidor serão as seguintes:

- **Cliente -> servidor:**

  Nome do arquivo de texto do qual deseja-se extrair as 10 palavras mais frequentes (ex. 'lorem.txt')

- **Servidor -> cliente:**

  (i) No caso em que o servidor tenha recebido um nome de arquivo de texto que não consta na pasta 'arquivos', localizada no mesmo diretório de servidor.py, a mensagem de erro enviada para o cliente será: 'ERRO - Arquivo nao encontrado!'

  (ii) No caso em que o servidor receba um nome de arquivo de texto válido (disponível na pasta 'arquivos'), a mensagem enviada para o cliente será uma string representando o array das 10 palavras mais frequentes, em ordem decrescente de frequência, no formato definido na Atividade 1.

2) A sequência de mensagens entre cliente e servidor será:

1º: Cliente envia o nome do arquivo de texto desejado para o servidor (requisição).

2º: Servidor responde à requisição do cliente com uma mensagem que pode ser de erro ou a lista das 10 palavras mais frequentes contidas no arquivo de texto solicitado pelo cliente.

3) No lado do servidor, o arquivo de texto é lido por meio do uso do método 'open' do Python, que recebe como parâmetros o caminho até o arquivo e o tipo de acesso que deve ser feito à ele (no caso, o tipo é 'rt' - read as text). O texto contido no arquivo é então transformado para caracteres minúsculos, de forma a padronizá-lo, e dividido por meio do método split em palavras, que são retornadas por esse mesmo método em um array. Esse array é passado como argumento para a instanciação de um Counter, uma subclasse da classe de dicionário em Python (dict) especialmente útil para este tipo de aplicação. O objeto do tipo Counter instanciado disponibiliza o método 'most_common', que recebe como parâmetro o número de elementos mais frequentes que deseja-se obter, ou seja, serve perfeitamente para o objetivo da aplicação. Em caso de erro de arquivo não encontrado durante a execução do método 'open', uma exceção do tipo FileNotFoundError é produzida, que foi então tratada no código de forma que, sempre que um arquivo não seja encontrado, uma mensagem de erro seja devolvida para o cliente.

O lado do cliente, por sua vez, é extremamente simples, já que todo o processamento, tratamento de erro e formatação de retorno é feito pelo servidor. O único ponto a se destacar é que a palavra-chave 'exit' pode ser digitada pelo usuário da aplicação no lugar do nome de um arquivo de texto para terminar a execução da aplicação cliente.