Tainá Lima
DRE: 116165607

Sistemas Distribuídos
Prof. Silvana Rossetto

Laboratório 2

Atividade 1

1. Objetivo

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

2. Camadas

1. Funcionalidades da camada de interface com o usuário: recebe do usuário o nome do arquivo e a palavra 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) o número de ocorrências da palavra no arquivo. **A camada de processamento retornará para a de interface somente o resultado da busca, ou seja, o número ocorrências da palavra ou um aviso da falta do arquivo com um esquema interno de retorno; de modo que é função da camada de interface tornar esse retorno mais "palatável" para o usuário.**

2. Funcionalidades da camada de processamento: solicita o acesso ao arquivo texto. Se o arquivo for válido, realiza a busca pela palavra informada e prepara a resposta para ser devolvida para a camada de interface. Se o arquivo for invalido, responde com a mensagem de erro. **O resultado será entregue para a camada de mensagem num esquema estruturado onde informará: (i) se encontrou ou não o arquivo; (ii) quantas ocorrências da palavra pedida existem; (iii) o nome do arquivo onde ocorreu a busca e a palavra buscada.**

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, devolve uma mensagem de erro.

Tainá Lima
DRE: 116165607

Sistemas Distribuídos
Prof. Silvana Rossetto

Atividade 2

1. 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.

2. Proposta de arquitetura de sistema:

a. Funções de cada papel

1. Lado cliente: implementa a **camada de interface e parte da de processamento para a transformação do dado resultante da busca em algo mais "palatável" para o usuário**. O usuário poderá solicitar o processamento de uma ou mais buscas em uma única execução da aplicação: o programa espera pelo nome do arquivo e da palavra de busca, faz o processamento, retorna o resultado, e então aguarda um novo pedido de arquivo e palavra ou o comando de finalização.

2. Lado servidor: implementa a **camada de processamento (a busca por palavra em arquivo)** e a **camada de acesso aos dados**. Projete 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). **O loop infinito é a situação default do servidor, contudo no código do servidor foi criado uma variável global "THRESHOLD" para caso se queira limitar o número de clientes atendidos para fins de teste.**

b. Fluxo de mensagens cliente-servidor

Inicialmente o lado cliente deve enviar uma **mensagem de pedido de busca** para o servidor que retorna primeiramente uma **mensagem de notificação de iniciação da busca**. Esta mensagem serve somente para notificar o cliente que a conexão foi aceita e, com o fato de que seu conteúdo é uma cópia do que foi recebido, é possível que o usuário verifique se os dados estão corretos. O cliente enviará a mensagem de pedido de busca de acordo com o padrão explicitado na seção seguinte e deverá ficar em espera pelo resultado após ter recebido a notificação de início da busca.
Ao finalizar seu serviço, o servidor retornará com a **mensagem de resposta**, informando se encontrou o arquivo ou não e quantas ocorrências da palavra existem; conforme a especificação de estrutura deste tipo de mensagem. Com isso, o lado cliente irá reescrever os dados numa saída

melhor entendível para o usuário, escrevendo-a na tela. Caso tenha ocorrido algum erro no processamento da busca, será retornada uma **mensagem de erro** ao invés de uma de resposta.
Após o retorno da mensagem de resposta/erro, o servidor ficará em espera por novas mensagens de pedido de busca. Caso o cliente queira finalizar a conexão com o servidor, ao invés de enviar uma mensagem de pedido de busca, deve enviar uma **mensagem de finalização**.

i. Estruturas das mensagens

	Função	Formatação	Tipo	Exemplo
Mensagem de pedido de busca	Pedir para que o servidor busque a palavra "word" dentro do arquivo de texto "namefile.extension". Nenhum dos campos mencionados pode ser vazio.	action:sch. file:name:<name file>. file:ext:<extension>. word:<word>.	String	action:sch. file:name:mercado. file:ext:csv. word:suco.
Mensagem de notificação de iniciação de busca	Avisar ao cliente que a busca foi iniciada, retornando exatamente o que recebeu do cliente.	action:sch. file:name:<name file>. file:ext:<extension>. word:<word>.	String	action:sch. file:name:mercado. file:ext:csv. word:suco.
Mensagem de resposta	Retorna em si o resultado da busca. Caso o arquivo não exista, o valor para o número de ocorrências é retornado como -1 e o campo "existFile" é False.	result:existFile:<True/False>. result:occr:<number>. result:file:<filename.extension>. result:word:<word>.	String	result:existFile:True. result:occr:5. result:file:mercado.csv. result:word:suco.
Mensagem de erro	Mensagem para avisar que houve algum problema com o processamento.	result:error.	String	result:error.
Mensagem de finalização	Finalizar a conexão entre o cliente e o servidor em específico, de forma que o cliente não seja	action:end.	String	action:end.

	fazer mais buscas, tornando o servidor disponível.			

ii. Tradução de mensagens

As mensagens trocadas entre o cliente e o servidor são todas no formato String, mas possuem uma estruturação interna semelhante a linguagem Turtle para RDF (Resource Description Framework) para facilitar o entendimento de ambas as partes.
A aplicação ao receber os dados do nome do arquivo e a palavra a ser buscada pelo usuário deve converter isso para o esquema de mensagem de pedido de busca. Para isso, é utilizado a função “encodeMessageCli” localizada no módulo “messageProtocol”. O mesmo ocorre quando o usuário deseja finalizar sua conexão com o servidor, teclando a palavra “end”.
Para o caminho inverso, ou seja, para extrair os dados da mensagem trocada, utiliza-se a função “decodeMessage” do mesmo módulo, armazenando-os numa estrutura de dicionário. No caso de necessitar mostrar tais dados para o usuário, transformando-o em informação, é necessário que se utilize a função “interpretMessage”, que retornará uma String cujo conteúdo é mais legível para o usuário.