Fluxo de execução

Nesse documento todo fluxo de execução do programa é detalhado, bem como suas funções, variáveis, entradas e saídas.

Funções

- *main*

Como próprio nome diz é a função principal da execução, é ela que faz a chamada de todas as outras funções necessárias. Inicialmente, fazendo uso de uma função da biblioteca boost, a "property_tree", ela realiza o parse do arquivo de configuração, que fica localizado na "pasta anterior" a do código, armazena todos os valores das tags em variáveis e os printa na tela para informar ao usuário quais parâmetros foram passados.

Após isso, inicia-se o *while* que vai garantir o loop de execução do programa. Dentro dele é chamada a função "findFiles" passando 7 argumentos, que são todos obtidos a partir do arquivo de configuração (no próximo tópico a função "findFiles" é detalhada). Na seqüência ele printa uma mensagem na tela e entra no *for* de 1 até o valor do timer, que representa quantos segundos a função "findFiles" levará para ser executada novamente. Dentro do *for* por meio de mais uma função da biblioteca boost é disparado um "sleep" de 1 segundo, que multiplicado pela quantidade de vezes que esse loop estiver ativo, resulta no tempo total. Como "escape" desse timer, se o usuário pressionar a tecla "q" o programa é encerrado.

Todo esse bloco de execução está dentro de uma estrutura de try-catch, printando na tela uma mensagem caso dê erro.

- *findFiles*

Essa função recebe como parâmetro 7 argumentos que são: uma string "origin" que é caminho de origem do arquivo, um string "destiny" que é o caminho para onde o arquivo vai ser transferido, uma string "fileName" com o nome do arquivo (ou prefixo ou extensão), um bool "regexSearch" que vale *true* ou *false* indicando se a busca deve ou não ser feita por regex, um string "regexValue" que é a própria regex, um bool "newestOldestSearch" que vale *true* ou *false* indicando se a busca deve ser feita usando o critério para encontrar o arquivo mais velho ou o mais novo e uma string "newestOldestValue" que vale *newest* para procurar o arquivo mais novo ou o*ldest* para procurar o arquivo mais velho.

De início são definidas algumas variáveis que serão usadas na execução da função: um bool "find" inicializado com *false*, uma regex "rx" que é inicializada com a conversão da variável "regexValue" para uma regex, dois vetores de string, "filePathDestiny" e "filePathOrigin", uma string "nomeAux", as strings "fileFullPathOrigin" e "fileFullPathDestiny", um bool "fileMatch", dois inteiros "pos" e "posExt" e um bool "findRegex".

Na sequência, um *for* é inicializado para "percorrer" todos os arquivos do diretório "origin", por meio do parâmetro "directory_iterator" presente na biblioteca "filesystem" que faz parte da boost. Dentro do *for*, alguns valores são atribuídos as variáveis, "nomeAux" recebe o nome

(com extensão) do arquivo que está sendo analisado no momento dentro do *for*, “fileMatch” recebe o valor inicial de *false*, fazendo uso da função *find*, que procura uma string dentro da outra, “pos” recebe o índice em que a string encontrada e se inicia dentro da string analisada e recebe o valor de -1 caso não encontre, para esse caso busca-se o “fileName” dentro de “nomeAux”. Segundo a mesma lógica, “posExt” recebe o índice de busca da string “.” dentro de “fileName” e por fim “findRegex” recebe o resultado da busca da regex “rx” dentro de “nomeAux”, por meio da função “regex_search”.

Após as atribuições, inicia-se a comparações lógicas, a mais externa verifica se a busca deve ou não ser feita a partir de regex, verificando o valor de “regexSearch”. Caso esse valor seja falso, outro *if* é utilizado para verificar a busca do arquivo a partir de uma dos três casos (nome completo, prefixo ou extensão), se alguns desses caso for verdadeiro, “fileMatch” recebe *true*. Caso o valor de “regexSearch” seja verdadeiro, é verificado se a variável “findRegex” também é verdadeiro, se for, “fileMatch” recebe *true.*

Continuando a lógica, se “fileMatch” for *true*, que indica que um arquivo foi encontrado, ocorrem novas atribuições, primeiro “fileFullPathOrigin” e “fileFullPathDestiny” recebem os valores de “origin” e “destiny”, respectivamente e depois são concatenados com os valores de uma string “\\” e com a conversão de “nomeAux” para string, sendo assim, “fileFullPathOrigin” nesse momento possui o caminho completo do arquivo de origem (path e nome do arquivo.extensão), o mesmo vale para “fileFullPathDestiny” mas com os valores de destino. Na sequência, “find” recebe *true*, indicando que encontrou um arquivo.

Após isso, é verificado a partir de “newestOldestSearch” se a busca deve ser feita com base na data de modificação dos arquivos, se for falso, é chamada a função “transferFindFiles”, que será explicada no próximo tópico, se o valor for verdadeiro, “fileFullPathOrigin” e “fileFullPathDestiny” são adicionados aos vetores “filePathOrigin” e “filePathDestiny, respectivamente, que na condição de busca pela data do arquivo, ao final do loop, os vetores vão possuir todos os caminhos de origem e destino.

Ao sair do *for*, é feita a verificação das variáveis “newestOldestSearch” e “find”, caso ambas sejam verdadeiras, a variável inteira “index” recebe o retorno da função “compareDate”, que será explicada mais a frente, e depois a função “transferFindFiles” é chamada passando como parâmetros os vetores “filePathOrigin” e “filePathDesitny” ambos na posição “index”. No final, verifica-se se houve pelo menos um arquivo encontrado dentro do diretório, caso não houver, a mensagem de “File(s) not found !” é printada na tela.

- *transferFiles*

Essa função é simples, ela recebe como parâmetro duas string, “origin” e “destiny” que possuem os caminhos completos dos arquivos (path e nome do arquivo) de origem e destino, respectivamente.

Dentro de uma estrutura try-catch, a função “copy_file”, que está presente dentro da bilioteca boost/filesystem, é chamada recebendo como argumentos “origin”, “destiny” e a opção

“overwrite_if_exists”, para que o arquivo seja sobre escrito caso ele já exista no diretório de destino. Essa função, copia o arquivo indicado de um diretório para o outro.

Feito isso, uma mensagem “File transfer with succes !” é printada na tela e a função “remove” também presente em boost/filesytem é chamada para remover o arquivo do diretório de origem.

Caso a execução falhe dentro do *try*, uma mensagem “Error transfer file” é printada na tela.

- *compareDate*

Esse função é do tipo inteiro e recebe como parâmetro um vetor de string “filePaths” e uma string “newestOldestValue”.

Inicialmente as variáveis utilizadas são declaradas: “now” e “timeEdit” do tipo time_t, com “now” recendo o valor de time(NULL) para obter a hora do momento de execução, “delta” e “maior” do tipo double e os inteiros “índex”, “maisNovo” e “maisVelho” todos inicializados com zero.

Na sequência, é inicializado um *for* para percorrer todos os elementos do vetor “filePaths”, a cada iteração a variável “timeEdit” recebe o valor do último horário de edição do arquivo, por meio da função “last_write_time” presente na biblioteca boost/filesystem. Após isso, as comparações são iniciadas, caso seja o primeiro elemento analisado, “maior” recebe o valor da diferença de horas entre “now” (hora do momento) e “timeEdit”, por meio da função “difftime” da biblioteca “ctime”. Caso não seja o primeiro elemento, “delta” recebe o valor dessa diferença e compara, se “delta” for menor que “maior” siginifica que o arquivo foi editado mais recentemente e por isso “maisNovo” recebe o valor de “index”, indicando a posição do caminho dentro do vetor. Porém, se “delta” for maior que “maior” significa que o arquivo foi editado há mais tempo e assim “maisVelho” recebe “index”. No final, “index” é incrementado.

Ao sair do *for*, é feita a comparação com base em “newestOldestValue”, caso seja igual a “newest”, significa que a busca deve ser feita para encontrar o arquivo editado mais recentemente e a função retorna o valor de “maisNovo”. Caso seja igual a “oldest”, significa a busca pelo arquivo mais velho e função retorna o valor de “maisVelho”.