EP3 - MAC0422 - 2015
main, file, directory, regular, command, block e stream

Renato Lui Geh e Ricardo Fonseca

Linha de input

Assim como nos últimos EPs, há uma string `cmd_line` que é a linha de input do usuário. Em seguida dividimos `cmd` em tokens.

Isso é feito na função `Tokenize`. Os tokens são armazenados numa tabela, sendo que o primeiro elemento equivale ao próprio comando, e não o primeiro argumento do comando.

Prompt e Comando

Foi usado `readline` e `history` para o prompt, assim como foi feito no EP1. Com o comando do usuário e seus argumentos, passamos a tabela com tais valores para a função `CommandToFunction` do módulo `utils`, que realizará, se possível, a função equivalente.

Atributos

A classe abstrata `file` é a representação de um arquivo em nosso programa. Ela possui os seguintes atributos:

`string name_`: Nome do arquivo

`time_t t_create_`: Tempo de criação do arquivo

`time_t t_modify_`: Tempo da última modificação do arquivo

`time_t t_access_`: Tempo do último acesso ao arquivo

Métodos

Além disso, file possui um método para mudar seu nome (`Rename()`), um para que calcula quantos blocos o arquivo ocupa (`long int Block()`) e 3 métodos que atualizam os tempos de criação, modificação e último acesso:

`RefreshCreationTime()`,

`RefreshModifiedTime()` e

`RefreshAccessedTime()`, respectivamente.

Existem também duas funções abstratas, `long int Size()` e `bool IsDirectory()`, que retornam o tamanho em bytes do arquivo e se essa classe na verdade é um diretório.

Operadores

Por último, damos override nos operadores `<`, `>` e `==`, para comparar arquivos. Escolhemos fazer as comparações lexicográficas, para depois facilitar a impressão em ordem alfabética.

Atributos

`directory` é uma classe derivada de `file`. Seus atributos são;:

`forward_list<File*> files_` : Lista dos arquivos dentro do diretório

`int n_files_` : Número de arquivos dentro do diretório

`long int files_sizeb_` : Tamanho em bytes dos arquivos dentro do diretorio

Métodos

A classe possue os seguintes métodos:
`ListFiles(FILE *stream)` : Imprime em stream os arquivos dentro do diretório.

`InsertFile(File *f)` : Insere o arquivo `f` nesse diretório.

`RemoveFile(string name)` : Remove o arquivo com nome `name` deste diretório.

`File* FindFile(string name)`: Procura e retorna o arquivo de nome `name`.

Além disso foram implementadas os métodos `long int Size` e `bool IsDirectory` da classe `file`.

Arquivo Regular

`regular` também é uma classe derivada de file, que representa um arquivo de puro texto. Possui como atributo `long int sizeb_`, que guarda o tamanho em bytes do arquivo, e métodos para ler e escrever seu conteúdo: `string ReadContent(FILE *stream)` e `WriteContent(string data)`.

Além disso `regular` da override nas funções de `Size` e `isDirectory` para retornar o tamanho do arquivo e falso, respectivamente.

Comandos

O módulo `command` possui um namespace com todos os commandos pedidos no enunciado, além de `SetPath(Directory dir)` que serve para setar o caminho atual do usuário.

Blocos

Nosso módulo `block` tem a implementação da classe de mesmo nome pra representar um bloco de espaço no disco.

Atributos

Ele possui três atributos:

`long int next_` : Índice do próximo bloco do arquivo.

`long int prev_` : Índice do bloco anterior do arquivo.

`long int index_`: Índice desse bloco.

`string content_`: Conteúdo em texto do bloco.

Métodos

A classe possui métodos para setar e retornar os índices (atuais quanto próximos/anteriores) e tamanho em bytes do conteúdo.

Além disso ele possui duas funções para escrever data:

`Write(string data)` : Sobrescreve o arquivo com `data`.

`Append(string data)`: Escreve no fim do arquivo `data`.

Stream

Esse módulo possui um namespace que organiza as entradas e saídas do programa.

Output

Namespace `Output` lida com as saidas para o sistema de arquivo com 4 métodos:

`Open(string filename)`: Abre um arquivo

`WriteMeta()`: Escreve a metadata do sistema de arquivos (o bitmap dos espaços livres, informações do FAT e Root).

`Write(Block *head)`: Escreve as informações de todos os blocos no FAT.

`Close()`: Fecha o arquivo.

Input

Namespace `Input` lida com as entradas do usuário e programa com 4 métodos:

`Open(string filename)`: Abre um arquivo

`ReadMeta()`: Lê a metadata do sistema de arquivos.

`Block* Read(long int index)`: Lê apenas o conteúdo de um bloco.

`Close()`: Fecha o arquivo.

Metadata

Namespace que tem atributos em pares (com a posição e tamanho) de metadata do sistema de arquivos:

`pair<long int, long int> kBitmapBlock`: Bitmap das posições livres de blocos (1 para ocupado, 0 para livre).

`pair<long int, long int> kFatBlock`: Bloco de Informações do FAT.

`pair<long int, long int> kRootBlock`: Bloco de informações do Root do sistema de arquivos.

`pair<long int, long int> kDiskBlock`: Blocos do restante do disco.

Exception

Por último, o namespace `Exception` serve para mensagens de erro, caso tenha um arquivo inválido ou algo tente ultrapassar o limite de 100MB do sistema de arquivos.

Observações Finais

Colocar aqui outras funções importantes que não mencionei, caso existam.

Observações

Não temos testes ainda :sadfaceemoji: