

Unix Based File System

Симеон Христов Христов, ФН: 72072, Курс: 3

Курсов проект по Структури от данни, проектът представлява имитация на файлова система с конзолен интерфейс работещ по подобие на **Unix**.

Компилиране и стартиране на програмата

В проектната папка ще намерите `Makefile` в който са описани необходимите файлове за компилация. За компилиране:

```
$ make build
```

След успешно компилиране трябва да стартирате генерираният файл. MacOS/ Linux:

```
$ ./generated_file_name.out
```

Windows:

```
$ start generated_file_name.exe
```

Структура на проекта

Проектът е разделен на 3 'модула' `file-system-domain` , `file-system-cli` , `file-system-utils` .

- `file-system-domain` - Модул съдържащ основната логика на програмата.
- `file-system-cli` - Модул отговарящ за обработването на подадените команди и подаването им до `file-system-domain` , отговаря също и за визуализиране на съобщения до потребителя.
- `file-system-utils` - Модул съдържащ помощни инструменти използвани от другите два модула.

`file-system-domain`

Модулът е разделен на 4 папки, `enums` , `file-management` , `model` , `utils` .

- `enums` - Тук се намира `enum FileType` който е използван за следене на типът на файловете по време на работа на програмата.
- `model` - Съдържа основните класове `модели` , които нямат никаква значима логика в себе си, а по-скоро имат ролята да 'моделират'данните в приложението.
- `file-management` - Съдържа класове работещи с `моделите` , тук се намира основната логика в програмата.
- `utils` - Съдържа помощни класове за улеснение на работата и подобрене на четимостта на кода на класовете във `file-management` .

`file-management`

Основните класове отговорни за логиката в приложението са `FileFactory` , `FileRepository` , `FileService` .

- `FileFactory` Клас отговорен за създаване на 'модели' по време на изпълнение на програмата.

```
class FileFactory
{
public:
    Directory *createDirectory(const std::string &name, Directory *parent);
    OrdinaryFile *createOrdinaryFile(const std::string &name, const std::string
&content, Directory *parent);
    SymbolicLink *createSymbolicLink(const std::string &name, const std::string
&filePath, Directory *parent);
};
```

- `FileRepository` Клас от ниския слой, отговорен за **съхранение, вмъкване, махане, промяна и намиране** на данни.

```
class FileRepository
{
private:
    Directory *root;
    FileFactory fileFactory;

public:
    void addDirectory(Directory *startingDirectory, const std::string filePath);
    void addFile(Directory *startingDirectory, const std::string &data, const
std::string &filePath, FileType fileType);
    void copyFile(Directory *startingDirectory, const std::string &fileToCopy, const
std::string &destination);
    File *find(Directory *startingDirectory, const std::string &filePath);
    void remove(const std::string filePath);
    Directory *getRoot();
};
```

- `FileService` Клас от средния слой в който се инжектира `FileRepository`, отговорен за по-сложната бизнес логика и сигнализиране за проблеми.

```
class FileService
{
private:
    FileRepository *repository;
    Directory *currentDirectory;
    DirectoryUtils directoryUtils;
    StringUtils stringUtils;

public:
    std::string getWorkingDirectory() const;
    // pwd
```

```

        std::string changeDirectory(const std::string &path);
// cd
        std::string getContentsList(const std::string &path) const;
// ls
        void concatenate(const std::vector<std::string> &filePaths, const std::string
&destinationFile); // cat
        void createOrdinaryFile(const std::string &content, const std::string
&destinationFile); // cat
        std::string getConcatenatedContents(const std::vector<std::string> &filePaths);
// cat
        void copyFiles(const std::vector<std::string> &filePaths, const std::string
&destinationPath); // cp
        void removeFile(const std::string &filePath);
// rm
        void makeDirectory(const std::string &filePath);
// mkdir
        void removeDirectory(const std::string &filePath);
// rmdir
        void makeSymbolicLink(const std::string &targetPath, const std::string
&linkLocation); // ln
        std::string getStat(const std::string &path) const;
// stat
};

```

model

Класовете отговорни за съхраняване и преставяне на данни са `MetaData` , `File` , `Directory` , `OrdinaryFile` и `SymbolicLink` .

- `MetaData` Структура за организация на **метаданните** за един файл.

```

struct MetaData
{
    int serialNumber;
    int fileSize;
    FileType fileType;
    unsigned lastAccessDate;
    unsigned lastContentModificationDate;
    unsigned lastMetaDataModificationDate;
};

```

- `File` Базов клас представляващ общите черти на всеки файл.

```

class File
{
protected:
    std::string name;
    MetaData metaData;

```

```

    File(const std::string &name, FileType fileType);

public:
    const std::string getName() const;
    const MetaData getMetaData() const;
    virtual const std::string getContent() const = 0;
    virtual File *getParent() const = 0;

    virtual void updateSize() = 0;
    void updateLastAccessDate();
    void updateLastContentModificationDate();
    void updateLastMetaDataModificationDate();
};

```

- `Directory` Модел наследяващ `File` който има способността да съхранява в себе си други модели наследници на `File`.

```

class Directory : public File
{
private:
    Directory *parent;
    std::vector<File *> subFiles;

public:

    void addFile(File *file);
    void removeFile(const std::string &name);
    const std::vector<File *> &getSubFiles() const;
    Directory *getParent() const;
    virtual const std::string getContent() const;
    virtual void updateSize();
};

```

- `OrdinaryFile` Модел наследяващ `File` който има способността да съхранява в себе си информация под формата на текст.

```

class OrdinaryFile : public File
{
private:
    Directory *parent;
    std::string content;

public:
    Directory *getParent() const;
    virtual const std::string getContent() const;
};

```

```
virtual void updateSize();  
};
```

- `SymbolicLink` Модел наследяващ `File` представляващ символна връзка към модел от тип `OrdinaryFile` позволяващ бърз достъп от всяко място във файловата система до файла към който е връзката.

```
class SymbolicLink : public File  
{  
private:  
    Directory *parent;  
    std::string filePath;  
  
public:  
    Directory *getParent() const;  
    virtual const std::string getContent() const;  
    virtual void updateSize();  
};
```

utils

Класовете имащи роля на помощни инструменти са `DirectoryUtils` , `IdGenerator` .

- `DirectoryUtils` Клас предоставящ инструменти за обхождане на директории, намиране и създаване на файлове.

```
class DirectoryUtils  
{  
private:  
    StringUtils stringUtils;  
  
    Directory *goUpTheHierarchy(Directory *directory) const;  
    Directory *goDownTheHierarchy(Directory *directory, const std::string  
&nextDirectoryName) const;  
    Directory *goToRoot(Directory *directory) const;  
    Directory *traverseDirectories(Directory *startingDirectory,  
std::vector<std::string> &pathSegments) const;  
    Directory *traverseAndCreateDirectories(Directory *startingDirectory,  
std::vector<std::string> &pathSegments) const;  
public:  
  
    std::string &getFullPath(Directory *directory) const;  
    Directory *findDirectory(Directory *startingDirectory, const std::string &path)  
const;  
    File *findFile(Directory * startingDirectory, const std::string &path) const;  
    void createDirectory(Directory * startingDirectory, const std::string &path)  
const;  
    void createFile(Directory *startingDirectory, const std::string &path, const
```

```
std::string &data, FileType type) const;
    OrdinaryFile *getFileFromSymLink(Directory *startingDirectory, const std::string
&path);
    std::string createFileCopyName(Directory *targetDirectory, const std::string
fileName);
};
```

- `IdGenerator` Клас с една инстанция, използван за номериране на файлове в системата.

```
IdGenerator IdGenerator::shared = IdGenerator();

class IdGenerator
{
public:
    static IdGenerator shared;
    long generateId();

private:
    long lastId;
    IdGenerator();
};
```

file-system-cli

Модулът е разделен на 3 папки `engine` , `input-handling` , `utils` .

- `engine` - Основната логика свързана с обработване на входни данни и изпращане до `FileService` класът.
- `input-handling` - Грижи се за прочитане и валидиране на данни въведени от потребителя.
- `utils` - Константи използвани от останалата част на модула.

engine

Съдържа класът `Engine` (най-горния слой). Той е отговорен за стартирането на програмата и обработването на грешки от `FileService` и `InputHandler` .

```
class Engine
{
private:
    std::string currentDirectoryName;
    FileService *fileService;
    InputHandler inputHandler;

public:
    void run();

private:
    void resolveInput(std::vector<std::string> &inputArguments);
```

```

int getCommandId(const std::string &command);

void printWorkingDirectory(); // pwd
void changeDirectory(std::vector<std::string> &inputArguments); // cd
void listFiles(std::vector<std::string> &inputArguments); // ls
void concatenateFiles(std::vector<std::string> &inputArguments); // cat
void copyFiles(std::vector<std::string> &inputArguments); // cp
void removeFiles(std::vector<std::string> &inputArguments); // rm
void makeDirectory(std::vector<std::string> &inputArguments); // mkdir
void removeDirectory(std::vector<std::string> &inputArguments); // rmdir
void makeSymbolicLink(std::vector<std::string> &inputArguments); // ln
void printStat(std::vector<std::string> &inputArguments); // stat

enum class Commands
{
    pwd,
    cd,
    ls,
    cat,
    cp,
    rm,
    mkdir,
    rmdir,
    ln,
    stat
};
};

```

input-handling

Съдържа `InputHandler` отговорен за прочитане и валидиране на входни данни от потребителя.

```

class InputHandler
{
private:
    StringUtils stringUtils;
    void validateCommand(const std::vector<std::string> &arguments);
    void validateArguments(const std::vector<std::string> &arguments);
    bool isValidArgument(const std::string &argument);
    void validateOutputRedirectOccurrenceAndPosition(const std::vector<std::string>
&arguments);

public:
    std::vector<std::string> readInput();
    std::string readInputForNewFile();
};

```

utils

Съдържа `cli_constants`, `namespace` от константи използвани във `file-system-cli`.

```
namespace cli {
    const std::string VALID_COMMANDS[] = {"pwd", "cd", "ls", "cat", "cp", "rm",
    "mkdir", "rmdir", "ln", "stat"};
    const int NUM_OF_VALID_COMMANDS = 10;
    const char INVALID_CHARACTERS[] = {'\\', '|', '`', '>'};
}
```

`file-system-cli`

Модулът съдържа `errors`, `meta_data_convert` и `StringUtils`.

`errors`

`namespace` от константи използвани във `file-system-cli` и `file-system-cli`.

```
namespace errors {
    const std::string NO_SUCH_FILE_OR_DIR = "ERROR: No such file or directory";
    const std::string FILE_NOT_DIRECTORY = ": File is not a directory.";
    const std::string FILE_DOES_NOT_EXIST = ": File does not exist.";
    const std::string FILE_ALREADY_EXISTS = ": File already exists.";
    const std::string FILE_NOT_ORDINARY = ": File is not an ordinary file";
    const std::string FILE_IS_DIRECTORY = ": File is a directory.";

    const std::string DIRECTORY_ALREADY_EXISTS = ": Directory already exists";
    const std::string DIRECTORY_DOES_NOT_EXIST = ": Directory does not exist.";
    const std::string FILE_IS_NOT_DIRECTORY = ": File is not a directory";
    const std::string CANT_DELETE_CURRENT_DIR = "ERROR: Can't delete current
directory";
    const std::string DIR_IS_NOT_EMPTY = ": Directory is not empty";

    const std::string NOT_SPECIFIED_FILE = "ERROR: No file name specified.";
    const std::string NOT_SPECIFIED_DESTINATION = "ERROR: No destination directory
name specified.";
    const std::string INVALID_PATH = ": Invalid path specified.";

    const std::string INVALID_COMMAND = ": Is not a valid command";
    const std::string MANY_OUTPUT_REDIRECTION_OPERATORS = "ERROR: There can be only
1 output redirection operator";
    const std::string MANY_ARGS_POST_OUTPUT_REDIRECTION_OPERATOR = "ERROR: Only one
argument is allowed after the output redirection operator";
    const std::string COMMAND_WITHOUT_OUTPUT_REDIRECTION = ": Command does not
require an output redirection operator";
    const std::string INVALID_NUMBER_OF_ARGUMENTS = ": Invalid number of arguments";
    const std::string ARGUMENT_WITH_INVALID_SYMBOLS = ": Argument contains invalid
symbols (`, \\, |, >)";
}
```


meta_data_convert

namespace с помощни функции за представяне на мета данни в по-удобен за четене формат

```
namespace metadata_convert {  
    std::string convertFileTypeToString(FileType fileType);  
    std::string convertUnixTimeStampToString(long timestamp);  
}
```

StringUtils

Клас с помощни функции за по-често срещаните и необходими операции със std::string

```
class StringUtils  
{  
public:  
    std::string removeWhiteSpaces(const std::string &input);  
    std::vector<std::string> segmentString(const std::string &input, char  
delimiter);  
    std::string getLastAfter(const std::string text, const std::string &delimiter);  
};
```

Демонстрация на използване

```
unix-file-system $ make build  
unix-file-system $ ./a.out  
/ $ pwd  
/  
/ $ mkdir directory1  
/ $ cat > directory1/file1  
This is a test  
input  
.  
/ $ ls  
directory1  
/ $ cd directory1  
directory1 $ ls  
file1  
directory1 $ cd ..  
/ $ cp directory1/file1 /  
/ $ ls  
directory1 file1  
/ $ cat file1  
This is a test  
input  
/ $ rm directory1/file1
```

```
/ $ ls directory1
```

```
/ $ stat directory1
```

```
Size: 0, Type: Directory, Last Modification: 12/1/2023 21:46:18, Last Access:  
12/1/2023 21:46:24, Last Meta Data Modification: 12/1/2023 21:46:24, SerialNo.:2
```

```
/ $ rmdir directory1
```

```
/ $ ls
```

```
file1
```

```
/ $
```