



**Katedra informatiky / PC**

Simulátor souborového systému

(Semestrální práce)

student: Štěpán Martínek  
studijní číslo: A15B0087P  
email: smartine@students.zcu.cz  
datum: 27. ledna 2017

# Obsah

<b>1</b>	<b>Zadání</b>	<b>1</b>
<b>2</b>	<b>Analýza úlohy</b>	<b>1</b>
<b>3</b>	<b>Popis implementace</b>	<b>2</b>
3.1	Datové struktury . . . . .	3
3.1.1	Array . . . . .	3
3.1.2	Node . . . . .	3
3.1.3	Childs . . . . .	3
3.2	Enumy a Makra . . . . .	4
3.2.1	bool . . . . .	4
3.2.2	returnCodes . . . . .	4
3.2.3	nullptr . . . . .	4
3.3	Soubory . . . . .	4
<b>4</b>	<b>Uživatelská příručka</b>	<b>5</b>
4.1	Překlad a spuštění . . . . .	5
4.1.1	Windows . . . . .	5
4.1.2	Linux . . . . .	6
4.1.3	Spuštění . . . . .	6
4.2	Dokumentace . . . . .	7
<b>5</b>	<b>Závěr</b>	<b>7</b>

# 1 Zadání

Naprogramujte v ANSI C přenositelnou **konzolovou aplikaci**, která načte ze souboru obraz souborového systému a seznam příkazů, které se mají nad souborovým systémem vykonat.

Program se bude spouštět příkazem: `fssim.exe (files) (commands)`. Symbol `(files)` zastupuje jméno souboru s obrazem souborového systému a symbol `(commands)` zastupuje jméno souboru s příkazy (popis vstupních souborů bude uveden dále). Váš program tedy může být během testování spuštěn například takto:

```
...\>fssim.exe files.txt commands.txt
```

Výstupem programu bude výstup jednotlivých příkazů vykonaných nad simulovaným souborovým systémem vypsáný do příkazové řádky. Pokud nebudou uvedeny právě dva argumenty, vypíše chybové hlášení a stručný návod k použití programu v angličtině podle běžných zvyklostí (viz např. ukázková semestrální práce na webu předmětu Programování v jazyce C). **Vstupem programu jsou pouze argumenty na příkazové řádce – interakce s uživatelem pomocí klávesnice či myši v průběhu práce programu se neočekává.**

Hotovou práci odevzdejte v jediném archivu typu ZIP prostřednictvím automatického odevzdávacího a validačního systému. Archiv necht' obsahuje všechny zdrojové soubory potřebné k přeložení programu, **makefile** pro Windows i Linux (pro překlad v Linuxu připravte soubor pojmenovaný **makefile** a pro Windows **makefile.win**) a dokumentaci ve formátu PDF vytvořenou v typografickém systému  $\text{\LaTeX}$ , resp.  $\text{\LaTeX}$ . Bude-li některá z částí chybět, kontrolní skript vaši práci odmítne.

## 2 Analýza úlohy

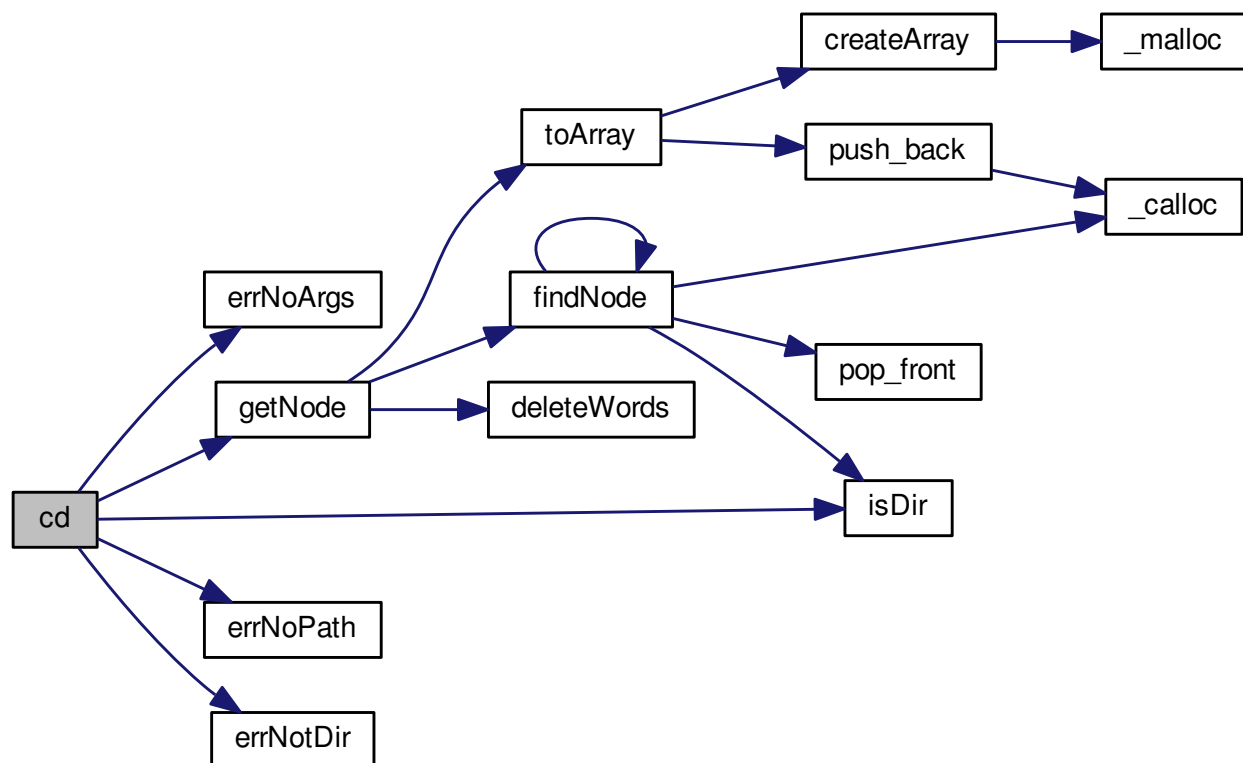
Pro souborový systém bude vhodné využít stromovou strukturu, kdy uzel (**Node**) bude mít v sobě ukazatel na rodiče a seznam potomků. Bude-li se jednat o kořen, bude mít jako rodiče nastaven sám sebe, aby se nemuselo ověřovat, zda rodič existuje, a simulovalo se linuxové chování.

Jediné statické pole bude jméno uzlu, které může být maximálně 256 znaků, ostatní pole budou dynamické. Pro pole budou využívány struktury obsahující velikost pole a ukazatel na samotné pole, výjimkou budou stringy, které budou mít vždy velikost `strlen+1`;

Pro alokaci paměti se budou používat wrappery, které ověří, zda byla paměť alokována, a případně ukončí program s příslušným exit kódem (viz uživatelská dokumentace).

### 3 Popis implementace

Normální dokumentaci lze na Linuxu vygenerovat přes `make doc`. (Potřebuje `doxygen` a `dot`). Dokumentace vygenerovaná `doxygenem` obsahuje i různé diagramy, jako například graf volání z funkce `cd` viz obrázek 1.



Obrázek 1: Hierarchie volání funkcí z commandu `cd`.

## 3.1 Datové struktury

### 3.1.1 Array

Datová struktura obsahující pole stringů a jeho velikost. Používá se obdobně jako kolekce deque, prvky se přidávají nakonec a odebírají ze začátku.

```
typedef struct array
{
    int size;
    char** string;
} Array;
```

### 3.1.2 Node

Datová struktura uzlu stromového filesystému. Obsahuje ukazatel na rodiče, který pokud se jedná o kořen je rodič nastaven na `nullptr`, a ukazatel na strukturu `Childs`.

```
struct node
{
    char name[256];
    struct node* parent;
    Childs* childs;
};
typedef struct node Node;
```

### 3.1.3 Childs

Datová struktura obsahující pole ukazatelů na uzly a velikost tohoto pole. Přes funkci `copyAndSortChilds` je možné udělat mělkou kopii struktury, která bude mít pole uzlů seřazené podle jména insert sortem.

```
typedef struct childs
{
    int size;
    struct node** arr;
} Childs;
```

## 3.2 Enumy a Makra

### 3.2.1 bool

Enum s hodnotami true a false pro přehlednější kód a snazší přechod od C++ k C.

```
typedef enum
{
    false = 0,
    true
} bool;
```

### 3.2.2 returnCodes

Enum s návratovými hodnotami programu.

```
enum returnCodes
{
    OK = 0,
    MALLOC_FAIL,
    WRONG_ARGUMENTS,
    NO_FILESYSTEM,
    NO_COMMANDS
};
```

### 3.2.3 nullptr

Makro umožňující přehlednější kód a umožňující snazší přechod od C++ k C.

```
#define nullptr 0
```

## 3.3 Soubory

**main.c:** Hlavní zdrojový soubor, obsahuje metodu **main** a načítání vstupních souborů.

**array.c array.h:** Obsahují strukturu **Array** a funkce pro manipulaci s ní.

**fssim.c fssim.h:** Obsahují funkce simulující příkazy filesystému.

**node.c node.h:** Obsahují struktury **Node** a **Childs** a funkce pro manipulaci s nimi.

`string.c` `string.h`: Obsahují funkce pro manipulaci s polem charů jako se stringem.

`util.c` `util.h`: Obsahují pomocné enumy, makra a wrappery funkcí `calloc` a `malloc`.

## 4 Uživatelská příručka

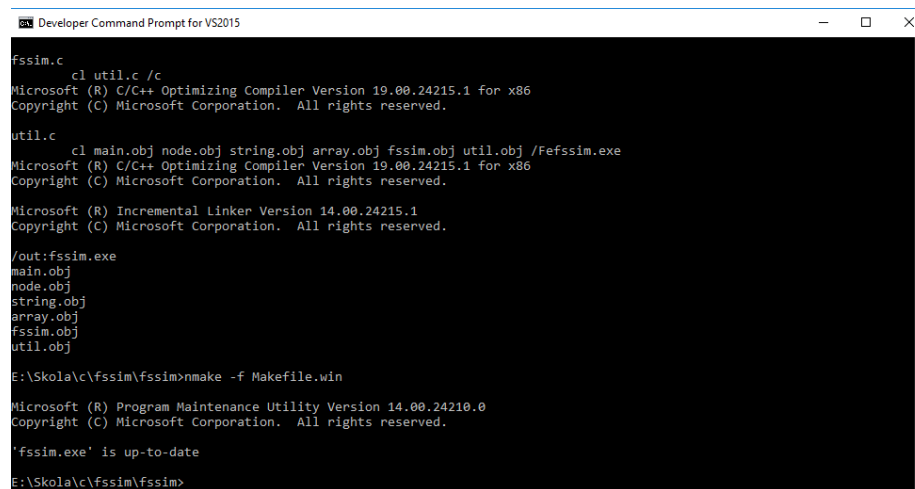
Program obsahuje soubory `Makefile` a `Makefile.win` pro snadnou kompilaci v Linuxu a Windows. Linuxová verze navíc umí spočítat řádky a vygenerovat dokumentaci.

### 4.1 Překlad a spuštění

#### 4.1.1 Windows

Spustíme Visual Studio Command Prompt. V něm pomocí příkazu `cd` vstoupíme do složky s programem `fssim` a příkazem `nmake -f Makefile.win` program zkompilujeme.

Příkazem `nmake -f Makefile.win clean` můžeme smazat veškeré soubory vytvořené kompilací.



```
Developer Command Prompt for VS2015
fssim.c
cl util.c /c
Microsoft (R) C/C++ Optimizing Compiler Version 19.00.24215.1 for x86
Copyright (C) Microsoft Corporation. All rights reserved.

util.c
cl main.obj node.obj string.obj array.obj fssim.obj util.obj /Fefssim.exe
Microsoft (R) C/C++ Optimizing Compiler Version 19.00.24215.1 for x86
Copyright (C) Microsoft Corporation. All rights reserved.

Microsoft (R) Incremental Linker Version 14.00.24215.1
Copyright (C) Microsoft Corporation. All rights reserved.

/out:fssim.exe
main.obj
node.obj
string.obj
array.obj
fssim.obj
util.obj

E:\Skola\c\fssim\fssim>nmake -f Makefile.win

Microsoft (R) Program Maintenance Utility Version 14.00.24210.0
Copyright (C) Microsoft Corporation. All rights reserved.

'fssim.exe' is up-to-date
E:\Skola\c\fssim\fssim>
```

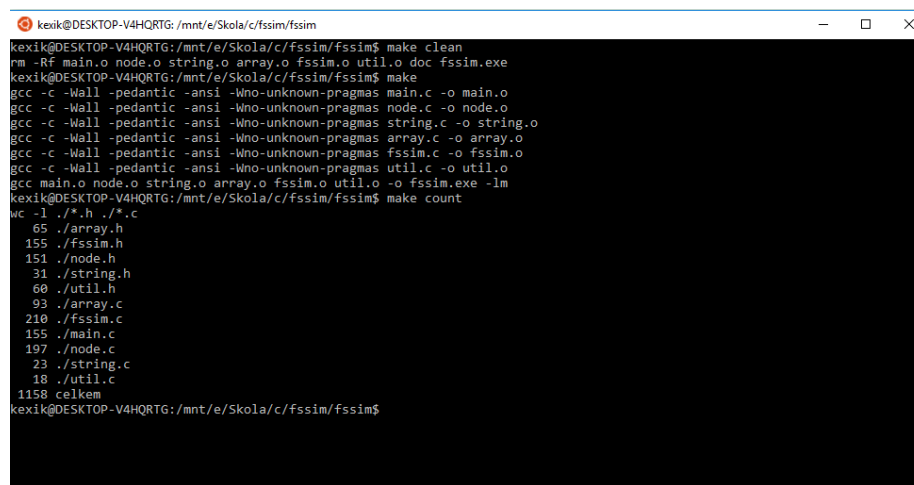
Obrázek 2: Kompilace v systému Windows.

### 4.1.2 Linux

Pomocí příkazu `cd` vstoupíme do složky s programem `fssim` a příkazem `make` program zkompilujeme.

Příkazem `make clean` můžeme smazat veškeré soubory vytvořené kompilací.

Příkazem `make count` můžeme vypsát počty řádků ve zdrojových souborech.



```
kezik@DESKTOP-V4HQRTG:/mnt/e/Skola/c/fssim/fssim$ make clean
rm -rf main.o node.o string.o array.o fssim.o util.o doc fssim.exe
kezik@DESKTOP-V4HQRTG:/mnt/e/Skola/c/fssim/fssim$ make
gcc -c -Wall -pedantic -ansi -Wno-unknown-pragmas main.c -o main.o
gcc -c -Wall -pedantic -ansi -Wno-unknown-pragmas node.c -o node.o
gcc -c -Wall -pedantic -ansi -Wno-unknown-pragmas string.c -o string.o
gcc -c -Wall -pedantic -ansi -Wno-unknown-pragmas array.c -o array.o
gcc -c -Wall -pedantic -ansi -Wno-unknown-pragmas fssim.c -o fssim.o
gcc -c -Wall -pedantic -ansi -Wno-unknown-pragmas util.c -o util.o
gcc main.o node.o string.o array.o fssim.o util.o -o fssim.exe -lm
kezik@DESKTOP-V4HQRTG:/mnt/e/Skola/c/fssim/fssim$ make count
wc -l /*.h /*.c
  65 ./array.h
 155 ./fssim.h
 151 ./node.h
  31 ./string.h
  60 ./util.h
  93 ./array.c
 210 ./fssim.c
 155 ./main.c
 197 ./node.c
  23 ./string.c
  18 ./util.c
1158 celkem
kezik@DESKTOP-V4HQRTG:/mnt/e/Skola/c/fssim/fssim$
```

Obrázek 3: Kompilace v systému Linux.

### 4.1.3 Spuštění

Zkompilovaný program se na Windows i Linuxu jmenuje `fssim.exe` a spouští se s argumenty `<filesystem> <commands>`, kde `<filesystem>` je jméno souboru s filesytémem obsahujícím absolutní cestu k jednomu adresáři/souboru na jedné řádce. Argument `<commands>` je jméno souboru s příkazy, který obsahuje jeden příkaz na řádce s jeho argumenty oddělenými mezerou.

Příklad spuštění na Linuxu: `./fssim.exe fs.txt cmd.txt`



## 4.2 Dokumentace

Pro vygenerování dokumentace na Linuxu jsou potřeba nástroje `doxygen` a `dot`. Jsou-li tyto nástroje nainstalovány, dokumentaci lze vygenerovat příkazem `make doc`.

Vygenerovaná dokumentace se nachází ve složce `doc` a při použití příkazu `make clean` je smazána.

## 5 Závěr

V jazyce C jsem již programoval, takže s ním jsem žádné problémy neměl. Aktivně programuji v jazyce C++ proto, jsem zvolil přístup vytvořit nejdříve funkční prototyp programu v C++ a poté ho přepsat do čistého C, který se ukázal jako velice vhodný. S programy `valgrind` `make` a `doxygen` jsem již měl také zkušenosti, takže ošetřit problémy s pamětí, vytvoření makefilu a dokumentace také nebyl problém.

Díky dobré struktulizaci programu stačily pro nově oběvená pravidla úpravy v rámci několika řádek. Největší úpravou bylo přidání nové porovnávací funkce. Při úpravách pro průchod validací se mi podařilo vytvořit memory leak, který se bohužel ve `valgrindu` s rozšířeným testovacím filesystémem maskoval jako leak vzniklý úplně jinde, což mě připravilo o spoustu času. Po použití Dr. Memory se mi podařilo nalézt pravou příčinu leaku a jednoduše jí opravit.

Osobně nesdílím názor, že C++ je natolik rozsáhlé, že by se nedalo zvládnout v jednom předmětu. Naopak si myslím, že PPA1 by mělo probíhat v C++ bez STL a PPA2 v C++ s STL.