

# Semestrální práce ZOS 2017

Práce se souborovým systémem pseudoNTFS

# Obsah

1.	Zadání práce .....	1
1.1	Seznam validních příkazů .....	1
2.	Parametry pseudo NTFS .....	2
3.	Struktura aplikace.....	2
3.1	Základní činnost aplikace.....	2
3.2	Jak aplikace pracuje s daty .....	2
4.	Spuštění programu .....	3
4.1	Stážení programu .....	3
4.2	Překlad programu .....	3
4.2	Spuštění programu .....	3
5.	Závěr .....	4

# 1. Zadání práce

Tématem semestrální práce je práce se souborovým systémem pseudoNTFS. Cílem je splnit několik vybraných úloh, které jsou podobněji specifikovány v dokumentech, na které je odkazováno níže.

Implementace probíhá v programovacím jazyce C.

**Aktuální zadání práce lze stáhnout na URL:**

<https://courseware.zcu.cz/CoursewarePortlets2/DownloadDokumentu?id=139572>

**Poznámky k zadání lze stáhnout na URL:**

<https://courseware.zcu.cz/CoursewarePortlets2/DownloadDokumentu?id=140060>

## 1.1 Seznam validních příkazů

- `cp s1 s2` Kopírování souborů
- `mv s1 s2` Přesun souborů
- `rm s1` Smazání souboru
- `mkdir a1` Vytvoří složku
- `rmdir a1` Smaže prázdný adresář
- `ls a1` Vypíše obsah adresáře
- `cat s1` Vypíše obsah souboru
- `cd a1` Změní aktuální cestu do adresáře
- `pwd` Vypíše aktuální cestu
- `info a1/s1` Vypíše informace o souboru
- `incp s1 s2` Nahraje soubor z pevného disku
- `outcp s1 s2` Nahraje soubor z pseudoNTFS
- `load s1` Vykoná sekvenci příkazů ze souboru
- `consist` Kontrola konzistence
- `defrag` Defragmentace
- `exit` Ukončení programu

## 2. Parametry pseudo NTFS

Jako parametry mé implementace pseudo souborového systému jsem zvolil:

- Počet clusterů 200
- Velikost clusteru 100
- Počet fragmentů pro jeden mft item 10
- Název domovské složky ROOT\_DIR
- ID domovské složky 0

## 3. Struktura aplikace

Pro implementaci jsem zvolil programovací jazyk C, kde jsem jednotlivé funkcionality dekomponoval do jednotlivých modulů. Tím jsem docílil větší přehlednosti zdrojového kódu.

Rozložení demonstruje následující tabulka:

Název souboru	Popisek
<b>boot_record</b>	Struktura pro uchování boot_recordu
<b>debugger</b>	Obsahuje kód speciálního nástroje pro debugging projektu
<b>functions</b>	Užitečné funkce, které pracují na vyšší úrovni (komplexnější)
<b>loader</b>	Načtení nebo vytvoření základního souboru se souborovým systémem
<b>main</b>	Hlavní třída aplikace, slouží ke spouštění
<b>mft</b>	Uchovávání MFTI a MFTF
<b>ntfs_helpers</b>	Nízkoúrovňové funkce pro práci s daty uloženými ve FS
<b>parametr</b>	Sdílená paměť využívaná při kontrole konzistence
<b>shell</b>	Zpracování jednotlivých příkazů z klávesnice
<b>shell_functions</b>	Obsahuje handlers pro zadávané příkazy

### 3.1 Základní činnost aplikace

Po spuštění programu *loader* ověří, jestli existuje souborový systém v požadovaném umístění, v případě, že neexistuje, tak dojde k jeho vytvoření s výchozími parametry, které jsou popsány v kapitole 2. Následně dojde k otevření souborového systému, jehož základní data se nahrají do paměti (*boot record*, *MFT tabulka*, *bitmapa*).

Následně je spuštěn *loader*, který čeká na příkazy zadávané s klávesnice. Po zadání příkazu se ověří existence příkazu a dojde případně ke spuštění příslušné části zdrojového kódu. V případě, že je zadán validní příkaz, tak *shell* spustí obslužný podprogram, který se nachází v modulu *shell\_functions*. Tyto obslužné podprogramy si pak následně použijí funkce z ostatních knihoven dle svých potřeb.

### 3.2 Jak aplikace pracuje s daty

Všechna data kromě obsahu clusterů si držím celou dobu aktuální v paměti a při manipulaci s daty nejprve provedu manipulaci na načtených datech a hned tyto změny zapisuji i do souboru. Tímto mám zajištěno, že při případném nekorektním ukončení práce s programem nedojde ke ztrátě dat.

## 4. Spuštění programu

### 4.1 Stažení programu

Aplikaci je možno stáhnout z repozitáře na GitHubu: <https://github.com/Sekiphp/KIV-ZOS>

### 4.2 Překlad programu

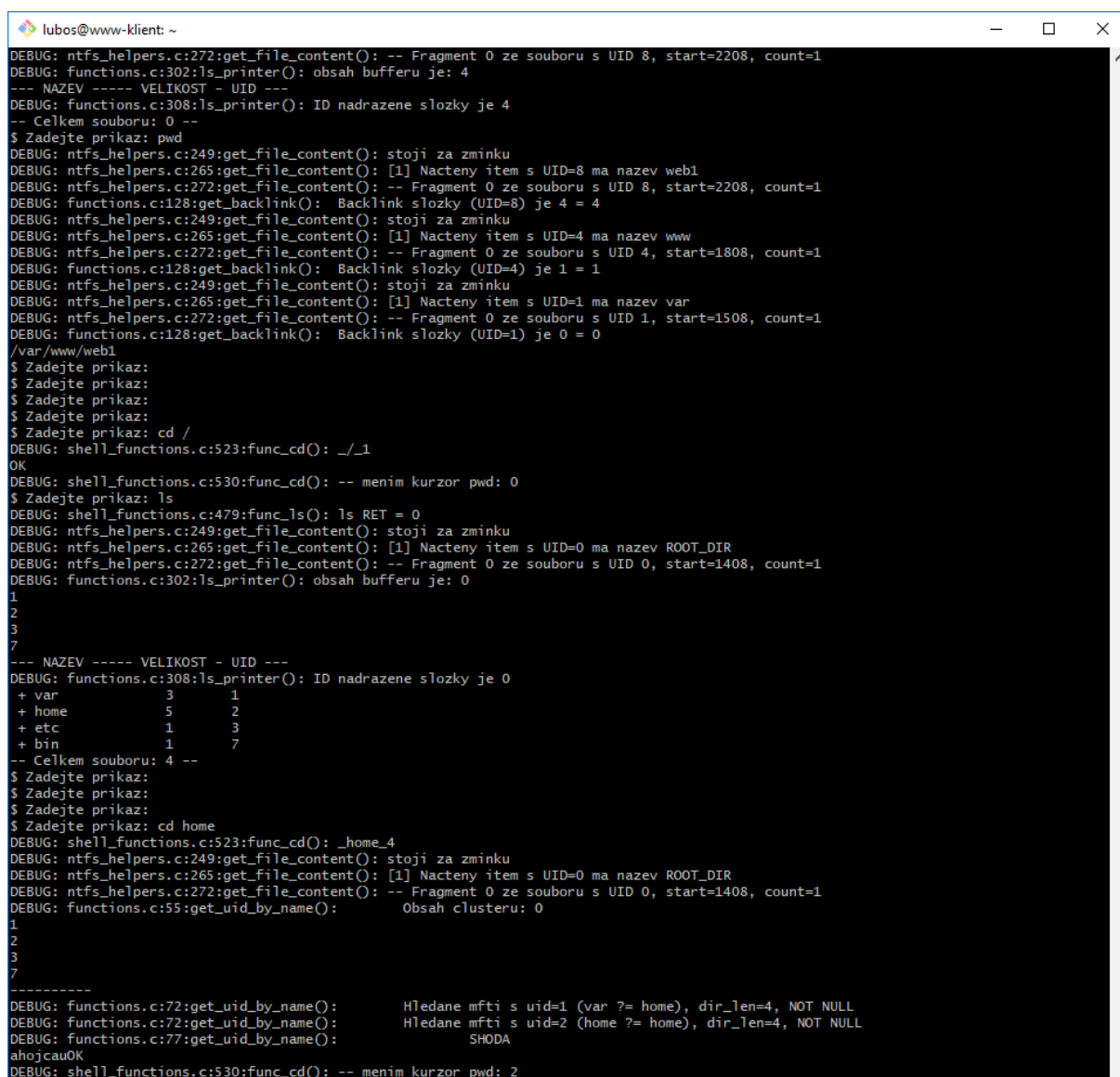
K přeložení programu dojde po zadání příkazu: **gcc \*.c -o pseudontfs -pthread -Wall**

Alternativním způsobem pro přeložení programu je využití utility *makefile* prostým zadáním příkazu: **make**

*(Poznámka: při použití příkazu make dojde bezprostředně po překladu ke spuštění aplikace)*

### 4.2 Spuštění programu

Pro spuštění programu je možné využít utility *make*, nebo zadat příkaz: **./pseudontfs ntfs.dat**



```
lubos@www-klient: ~
DEBUG: ntfs_helpers.c:272:get_file_content(): -- Fragment 0 ze souboru s UID 8, start=2208, count=1
DEBUG: functions.c:302:ls_printer(): obsah bufferu je: 4
--- NAZEV ----- VELIKOST - UID ---
DEBUG: functions.c:308:ls_printer(): ID nadrazene složky je 4
-- Celkem souboru: 0 --
$ Zadejte prikaz: pwd
DEBUG: ntfs_helpers.c:249:get_file_content(): stojí za zminku
DEBUG: ntfs_helpers.c:265:get_file_content(): [1] Nacteny item s UID=8 ma nazev web1
DEBUG: ntfs_helpers.c:272:get_file_content(): -- Fragment 0 ze souboru s UID 8, start=2208, count=1
DEBUG: functions.c:128:get_backlink(): Backlink složky (UID=8) je 4 = 4
DEBUG: ntfs_helpers.c:249:get_file_content(): stojí za zminku
DEBUG: ntfs_helpers.c:265:get_file_content(): [1] Nacteny item s UID=4 ma nazev www
DEBUG: ntfs_helpers.c:272:get_file_content(): -- Fragment 0 ze souboru s UID 4, start=1808, count=1
DEBUG: functions.c:128:get_backlink(): Backlink složky (UID=4) je 1 = 1
DEBUG: ntfs_helpers.c:249:get_file_content(): stojí za zminku
DEBUG: ntfs_helpers.c:265:get_file_content(): [1] Nacteny item s UID=1 ma nazev var
DEBUG: ntfs_helpers.c:272:get_file_content(): -- Fragment 0 ze souboru s UID 1, start=1508, count=1
DEBUG: functions.c:128:get_backlink(): Backlink složky (UID=1) je 0 = 0
/var/www/web1
$ Zadejte prikaz:
$ Zadejte prikaz:
$ Zadejte prikaz:
$ Zadejte prikaz:
$ Zadejte prikaz: cd /
DEBUG: shell_functions.c:523:func_cd(): _/_1
OK
DEBUG: shell_functions.c:530:func_cd(): -- menim kurzor pwd: 0
$ Zadejte prikaz: ls
DEBUG: shell_functions.c:479:func_ls(): ls RET = 0
DEBUG: ntfs_helpers.c:249:get_file_content(): stojí za zminku
DEBUG: ntfs_helpers.c:265:get_file_content(): [1] Nacteny item s UID=0 ma nazev ROOT_DIR
DEBUG: ntfs_helpers.c:272:get_file_content(): -- Fragment 0 ze souboru s UID 0, start=1408, count=1
DEBUG: functions.c:302:ls_printer(): obsah bufferu je: 0
1
2
3
7
--- NAZEV ----- VELIKOST - UID ---
DEBUG: functions.c:308:ls_printer(): ID nadrazene složky je 0
+ var          3      1
+ home         5      2
+ etc          1      3
+ bin          1      7
-- Celkem souboru: 4 --
$ Zadejte prikaz:
$ Zadejte prikaz:
$ Zadejte prikaz:
$ Zadejte prikaz: cd home
DEBUG: shell_functions.c:523:func_cd(): _home_4
DEBUG: ntfs_helpers.c:249:get_file_content(): stojí za zminku
DEBUG: ntfs_helpers.c:265:get_file_content(): [1] Nacteny item s UID=0 ma nazev ROOT_DIR
DEBUG: ntfs_helpers.c:272:get_file_content(): -- Fragment 0 ze souboru s UID 0, start=1408, count=1
DEBUG: functions.c:55:get_uid_by_name(): Obsah clusteru: 0
1
2
3
7
-----
DEBUG: functions.c:72:get_uid_by_name(): Hledane mfti s uid=1 (var != home), dir_len=4, NOT NULL
DEBUG: functions.c:72:get_uid_by_name(): Hledane mfti s uid=2 (home != home), dir_len=4, NOT NULL
DEBUG: functions.c:77:get_uid_by_name(): SHODA
ahojcawOK
DEBUG: shell_functions.c:530:func_cd(): -- menim kurzor pwd: 2
```

Obrázek 1: Ukázka spuštění programu v debug režimu

## 5. Závěr

Podařilo se mi implementovat poměrně komplexní celek, který reprezentuje virtuální souborový systém NTFS. Výsledný program jsem testoval a vyvíjel na Linuxu, jelikož nativní podporu pro OS Windows jsem nezamýšlel. Nejspíš by ale netrvalo mnoho času a tato podpora by se dala poměrně snadno doimplementovat.

Realizace práce byla o dost delší, než jsem si původně myslel. Veškerý svůj čas strávený s implementací práce jsem si pečlivě měřil a dostal jsem se na **86 hodin**.

Aplikace splňuje všechny prvky kladené zadáním.