FAKULTA
APLIKOVANÝCH VĚD
ZÁPADOČESKÉ
UNIVERZITY
V PLZNI

Simulace operačního systému

Semestrální práce z předmětu KIV/OS

Linha, Michal
mlinha@students.zcu.cz
A19N0036P

Danišík, Vojtěch
danisik@students.zcu.cz
A19N0028P

Čarnogurský, Jan
cagy@students.zcu.cz
A19N0025P

6. prosince 2019

Obsah

Kapitola 1

Zadání

- Vytvořte virtuální stroj, který bude simulovat OS
- Součástí bude shell s gramatikou cmd, tj. včetně exit
- Vytvoříte ekvivalenty standardních příkazů a programů
  - echo, cd, dir, md, rd, type, find /v /c""(tj. co dělá wc v unix-like prostředí), sort, tasklist, shutdown
    - cd musí umět relativní cesty
    - echo musí umět @echo on a off
    - type musí umět vypsat jak stdin, tak musí umět vypsat soubor
  - Dále vytvoříte programy rgen a freq
  - rgen bude vypisovat náhodně vygenerovaná čísla v plovoucí čárce na stdout, dokud mu nepřijde znak Ctrl+Z //EOF
  - freq bude číst z stdin a sestaví frekvenční tabulku bytů, kterou pak vypíše pro všechny byty s frekvencí větší než 0 ve formátu: "0x%hhx : %d"
- Implementujte roury a přesměrování
- Nebudete přistupovat na souborový systém, ale použijete simulovaný disk Za 5 bonusových bodů můžete k realizaci souborového systému použít semestrální práci z KIV/ZOS - tj. implementace FAT.

Při zpracování tohoto zadání použijte a dále pracujte s kostrou tohoto řešení, kterou najdete v archívu os_simulator.zip. Součástí archívu, ve složce compiled, je soubory checker.exe a test.exe. Soubor checker.exe je validátor semestrálních prací. Soubor test.exe generuje možný testovací vstup pro vaši semestrální práci.

Vaše vypracování si před odevzdáním zkontrolujte programem checker.exe. V souboru checker.ini si upravte položku Setup_Environment_Command, v sekci General, tak, aby obsahovala cestu dle vaší instalace Visual Studia. Např. vzorové odevzdání otestujete příkazem "compiled\vzorove odevzdani.zip", spuštěného v kořenovém adresáři rozbaleného archívu. Odevzdávaný archív nemá obsahovat žádné soubory navíc a program musí úspěšně proběhnout.

Kapitola 2

Analýza problému

2.1 Kernel

Kernel, neboli jádro systému, je program, který umožňuje spouštět procesy nebo vlákna, provádět vstupní a výstupní operace, zpracovávat přerušení a nebo spravovat ostatní zařízení. Má vyhrazený prostor paměti a dále úplnou kontrolu nad systémem. Jádro může být monolitické, mikrojádro, nebo hybridní, které spojuje obě možnosti dohromady. Monolitické jádro obsahuje všechny služby operačního systému, zatímco mikrojádro obsahuje jen ty nejdůležitější části, ostatní běží v uživatelském prostoru paměti.

2.1.1 Procesy

Proces je běžící instance programu a záznamy o procesu, které se nazývají PCB, se ukládají do tabulky procesů v jádře. Záznam PCB obsahuje například:

- PID (Process ID)
- Adresní prostor
- Hodnoty registrů

- Odkaz na další záznam
- Priorita procesu
- Soubory

Základními stavy procesů jsou vytvořený, do kterého se proces dostane po jeho vytvoření dále následuje připravený, kdy proces čeká na spuštění, běžící a ukončený. Dále se může dostat do stavu blokovaný nebo připravený a pozastavený, pokud byl zablokován nějakou IO akcí, nebo postaven plánovačem. Stavy lze vidět na obrázku 2.1.

Obrázek 2.1: Graf stavů procesů.

2.1.2 Vlákna

Vlákno je ve většině systémů součástí procesu a je to jeho nejmenší sekvence instrukcí. Podobně jako proces, i vlákno obsahuje strukturu pro ukládání informací o vláknech. Tato struktura se jmenuje TCB a obsahuje mimo jiné:

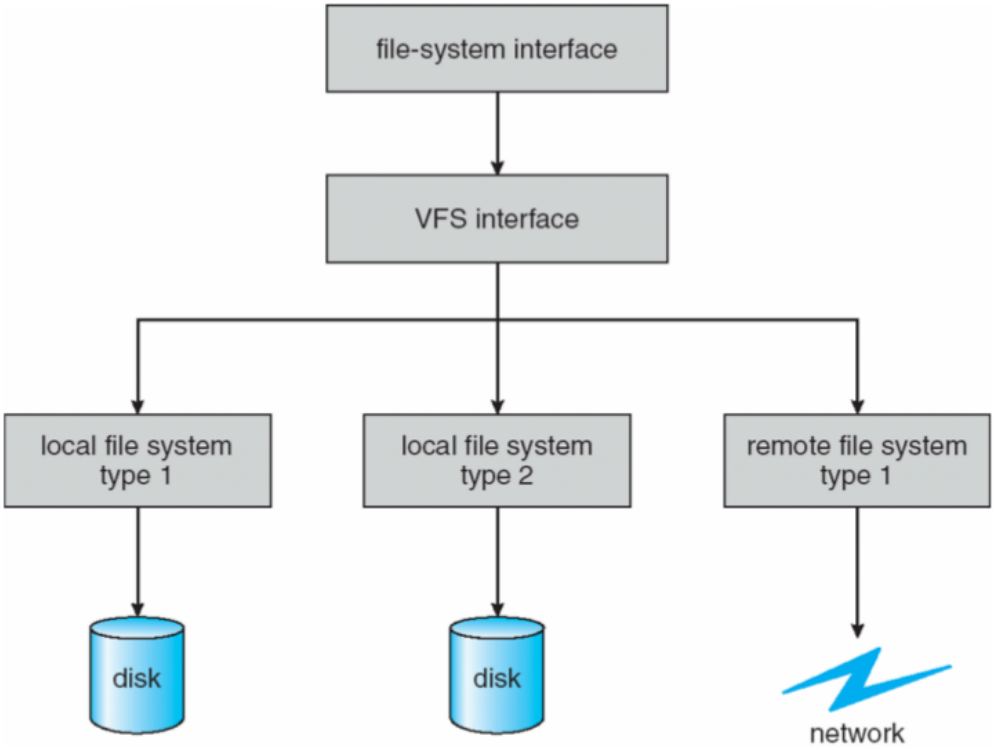
- TID (Thread ID)
- Ukazatel na zásobník
- Ukazatel na PCB pod který vlákno patří
- Stav vlákna

2.2 IO

IO, neboli vstupní a výstupní operace, slouží pro práci se soubory. Dále pak pro práci s hardware, který potřebuje nějaký vstup nebo generuje výstup, jako jsou klávesnice, myši apod. Další možností je pak grafický vstup a výstup na obrazovce.

2.2.1 Souborový systém

Souborový systém slouží pro správu dat uložených na úložišti. Skládá se z několika částí a to virtuálního souborového systému (VFS), dále pak z rozhraní mezi VFS a operačním systémem a samotné implementace souborového systému. Existuje několik druhů souborových systémů. Mezi hlavní patří NTFS, FAT a ext. Složení lze vidět na obrázku 2.2.



Obrázek 2.2: Struktura souborového systému.

2.2.2 Roury a přesměrování

Roury v operačních systémech představují meziprocesorovou komunikaci, kde oba procesy jsou spuštěny současně. Výstup prvního procesu je poté použit jako vstup druhého procesu, jde tedy o jednosměrnou komunikaci. Rouru může reprezentovat část paměti, kterou lze nazývat virtuálním souborem, do kterého jeden proces vkládá data a druhý je čte.

Přesměrování je způsob práce se vstupem a výstupem procesu, při kterém se nepoužije standardní vstup nebo výstup, ale použije se například vstup ze souboru.

2.2.3 Shell

Shell je program umožňující uživateli zadávat příkazy z konzole. Shell se stará o načtení a kontrolu správnosti vstupu a také o provedení příkazů.

Kapitola 3

Popis implementace

3.1 Procesy a vlákna

Procesy jsou v simulaci operačního systému reprezentovány třídou `Process`, která je definována v jádře v souboru `process.h`, a jedná se vlastně o záznam PCB. Tato třída obsahuje atribut `process_ID`, který reprezentuje PID procesu. Dále obsahuje atributy `process_thread_ID`, který ukládá ID vlákna hlavního procesu, `state`, což je stav procesu, `name` pro jméno procesu, `working_dir`, ukládající pracovní adresář procesu, `threads`, tedy mapu vláken procesu a nakonec handly pro vstup a výstup. Vytvořené instance tříd jsou ukládány do tabulky procesů, která je atributem třídy `IO_process`.

`IO_process` je třída, která se stará o práci s procesy, tedy o jejich vytváření, mazání, nebo úpravu. Po vytvoření procesu je procesu nastaven stav Runnable. Po spuštění hlavního vlákna s procesem se proces přesune do stavu Running. Po ukončení se procesu nastaví stav Exited.

`IO_process` kromě tabulky procesů obsahuje i mapu, která spojuje ID vlákna a ID procesu a dále mapu spojující ID procesu nebo vlákna s ID handlu. Pro práci s procesy obsahuje metody pro klonování procesů nebo vláken, a nebo metody pro jejich synchronizaci a to `Wait_For()`, `Notify()` a `Notify_All()`. Metoda Wait_For() pro svojí činnost potřebuje pole handlů procesů nebo vláken na které se má čekat. Poté je pole procházeno dokud se první z vložených procesů nedostane do stavu Exited, tedy neskončí, a tím je průchod ukončen

a do registru ax je uložen handle ukončeného procesu.

Vlákna jsou reprezentována v souboru thread.h třídou Thread. Třída Thread obsahuje několik atributů. Atributy `thread_ID` a `parent_ID` představují identifikátory vlákna a rodičovského vlákna. Dále pak atribut state reprezentuje stav vlákna. Atribut `std_thread` obsahuje odkaz na vlákno ze standardních knihoven jazyka C++. Dalšími atributy jsou registers, což jsou registy vlákna, `exit_code`, což je návratová hodnota, a entry_point, který obsahuje vstupní bod vlákna a je typu `TThread_Proc` z `api.h`. Dále vlákno obsahuje několik map a to mapu handlerů pro ukončení `terminate_handlers`, mapu čekajících vláken `handlers_waiting_for` a mapu uspaných handlerů sleeped_handlers. Posledními atributy jsou podmínková proměnná, `cv`, `mutex` a atribut udávající ID procesu, který byl probuzen handlerem, `waked_by_handler`.

Třída Thread obsahuje konstruktor pro vytvoření vlákna, kde se jeho stav nastaví na Runnable (konstruktor pro jádro nastaví stav na Running), metodu `Start()` pro spuštění vlákna a nastavení stavu na Running, metodu `Join()`, která přesune vlákno do stavu Exited. Metoda Stop() pak převede do stavu Blocked, pokud je vlákno ve stavu Running a pokud existuje 1 a více čekajících vláken. `Remove_Handler_From_Handlers_Waiting_For()` vlákno znovu spustí.

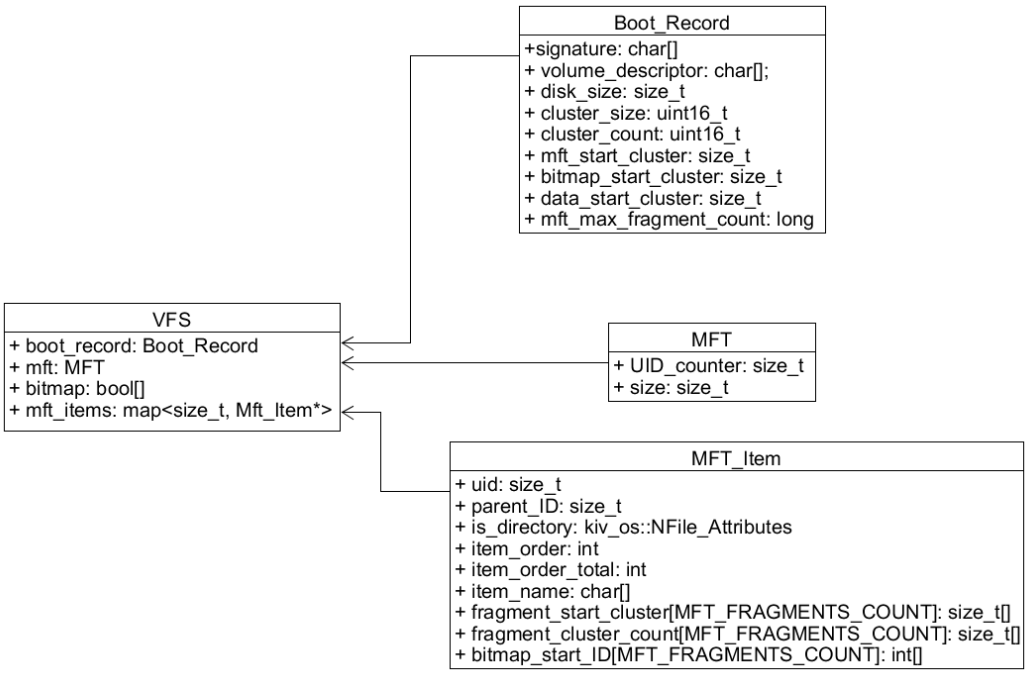
3.2 IO

3.2.1 Souborový systém

Jako souborový systém byl vybrán PseudoNTFS, který byl vytvořen v rámci předmětu ZOS. Pro jeho použití pro semestrální práci ho ale bylo nutné upravit. Souborový systém obsahuje několik částí. Mezi ně patří Boot Record, MFT, bitmapu a ID disku.

Boot Record je část, která obsahuje základní informace o souborovém systému. Obsahuje tedy informace o velikosti disku, velikosti clusterů, počtu clusterů, cluster počátku MFT, cluster počátku bitmapy a cluster počátku dat.

MFT (Master File Table) je tabulka skládající se z `MFT_Itemů`, jejich počtu a proměnná reprezentující prvního volného identifikátoru. Má určenou velikost a počet UID. `MFT_Item` je položka, která ukládá informace o souboru nebo složce. Udržuje UID položky, ID rodiče, atribut, zda je soubor složka, jméno položky a velikost položky v bytech. Dále obsahuje také `Exists_Item`, což je pomocný typ použitý při kontrole cesty. Celá MFT zabírá 10% velikosti disku. Diagram souborového systému lze vidět na obrázku 3.1.



Obrázek 3.1: Diagram VFS.

3.2.2 IO funkce

Pro práci se vstupem a výstupem slouží několik funkcí vytvořených v knihovně RTL. Dále jsou ve třídě `Functions` metody, které slouží jako rozhraní mezi VFS a jádrem.

Otevření souboru probíhá pomocí funkce `Open_File()` tak, že po zavolání příslušné funkce, dojde k pokusu o otevření souboru. Do funkce se nastavují vlajky a atributy souboru. Pokud je nastavena hodnota vlajky na `fmOpen_Always`, musí existovat soubor, který se systém pokouší otevřít.

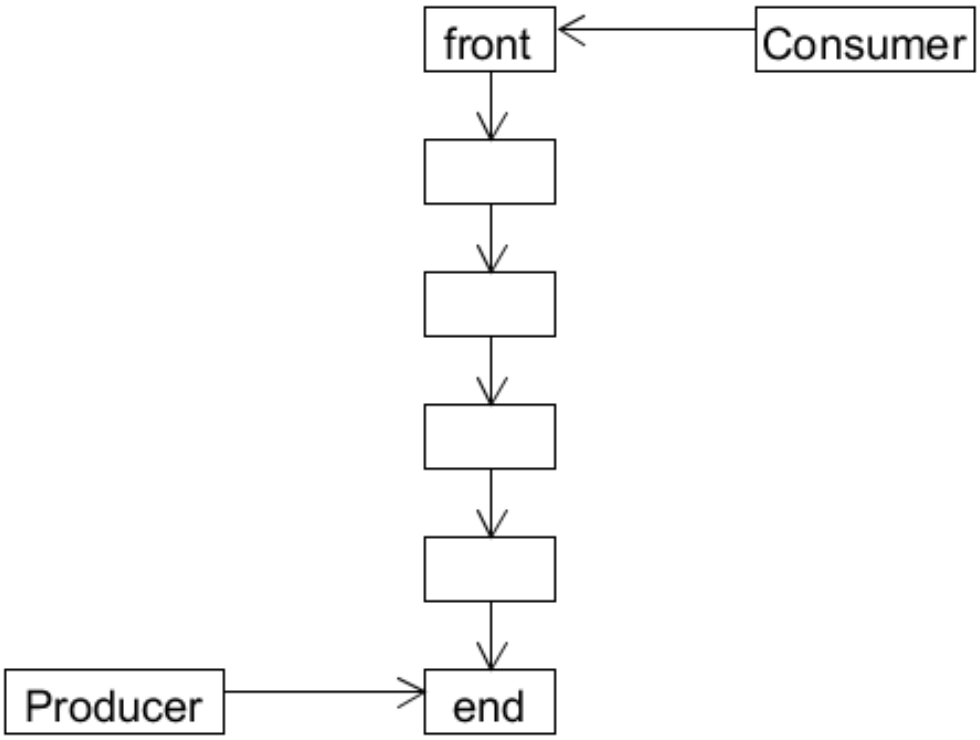
Po správném otevření souboru, je nastaven handle souboru z registrů po systémovém volání pro otevření souboru. Při systémovém volání jsou poté volány metody ze souborového systému buď pro získání `MFT_Itemu`, nebo pro jeho vytvoření. Zavření souboru je provedeno zavřením handlu pomocí příslušné metody, odstranění pak pomocí funkce `Delete_File()`, která opět provede systémové volání, které použije metodu ze souborového systému pro odstranění `MFT_Itemu`.

Pro čtení se používá funkce `Read_File()`. Pro svojí funkčnost potřebuje buffer, do kterého se budou data načítat, jeho velikost a proměnnou pro vrácení velikosti přečtených dat. Funkce provede systémové volání pro čtení, pomocí kterého se zavolá obsluha čtení ze souboru, která je definována v souboru `io_handle.cpp`, kde je provedeno čtení a práce se souborovým systémem. Obdobně funguje i zápis.

`Seek()` slouží pro přesun pozice, od které se má ze vstupu číst. Funkce, která toto provádí potřebuje informaci, z jaké pozice má číst, od jakého indexu se počítá začátek aktuální pozice a velikost souboru.

3.2.3 Roury a přesměrování

Pro práci s rourami slouží funkce `Create_Pipe()` z knihovny RTL, která vyžaduje dva handly, jeden pro vstup do roury a jeden pro výstup z ní. Samotná funkcionalita roury byla implementována pomocí dvou semaforů a mutexu, které rouru řeší jako problém producenta a konzumenta, a lze ji nalézt v souboru `pipe.cpp` v jádře. Rouru reprezentuje třída `Pipe`, která kromě semaforů a mutexu dále také obsahuje buffer, který je typu list `harů`. Pro zapisování do bufferu roury slouží metoda `Produce()`, kde dochází k přidávání dat na konec listu, pro čtení dat z roury pak metoda `Consume()`, ve které se vybírají data ze začátku listu. Zápis a čtení z roury je prováděno po jednotlivých `charech`, tedy bytech. Pro uzavření roury slouží metoda `Close()`. Struktura roury je zobrazena na obrázku 3.2.



Obrázek 3.2: Struktura roury.

V práci bylo implementováno jak přesměrování vstupu ze souboru tak vstupu do souboru. Při přesměrování, je při spouštění programu zjištěno, zda se jedná o přesměrování a o jaké. Poté dojde k otevření souboru, kde pokud se jedná o přesměrování vstupu, je souboru při otevírání nastavena vlajka, že musí pro otevření existovat, jinak není otevřen. Poté je v případě přesměrování vstupu, nastaven handle vstupu procesu na handle souboru a výstup na standardní výpis do konzole. V případě přesměrování výstupu je výstupní handle nastaven na handle souboru a handle vstupu na standardní vstup do konzole. Pokud se nejedná o přesměrování je vstupní handle nastaven na standardní vstup a výstupní handle na standardní výstup.

3.3 Programy

3.3.1 Parser a spouštění programů

Po načtení dat z *shell*, je nutné tato data nejprve zpracovat. To zařizuje command parser, který z dat získá jednotlivé příkazy. Z načteného bufferu se vytvoří stringstream, pomocí kterého jsou zadaná data rozdělena na tokeny. První token je vždy použit jako základ příkazu, tedy příkaz samotný. Poté jsou načítány argumenty do té doby, než je token roura nebo přesměrování. Příkaz a jeho argumenty jsou uloženy do třídy `Command`. V případě přesměrování je instanci třídy nastavena vlajka, že se jedná o přesměrování a o jaké. Zbytek vstupu za znakem přesměrování je použit jako název souboru a opět uložen do instance. V případě roury je opět použit první token za znakem roury jako další příkaz a další tokeny jako jeho argumenty. To vše se opakuje dokud se nepoužijí všechny tokeny.

Po provedení parsování je nutné načtený příkaz nebo příkazy spustit. Toto provádí command executor, který jednotlivé příkazy spustí a poté počká na jejich dokončení.

3.3.2 rgen

Jedná se o program, který po spuštění generuje náhodná čísla v plovoucí čárce na výstup. Generování náhodných čísel je implementováno v cyklu `while`, který běží do té doby, dokud uživatel nezadá na vstup hodnotu EOF, tedy klávesovou kombinaci CTRL + Z nebo nestiskne klávesu ENTER. Implementace je rozdělena do dvou vláken, kde jedno vlákno provádí samotné generování náhodných čísel a druhé kontroluje hodnotu na vstupu. Pro generování náhodných čísel byla použita funkce rand(). Při každém spuštění programu *rgen* se vygeneruje hodnota, podle které se náhodné hodnoty generují, pomocí funkce `seed()`. Před spuštěním cyklu generování je nejprve z hlavního vlákna spuštěno vlákno pro kontrolu vstupu, kterému je pomocí registrů předán ukazatel na `boolean` hodnotu, která signalizuje, že uživatel ukončil generování. Tato proměnná se jmenuje `eof`. Toto vlákno čte cyklicky ze vstupu a pokud hodnota počtu přečtených bytů bude 0, cyklus se ukončí a dojde k nastavení hodnoty proměnné `eof` na `true` a poté dojde k ukončení vlákna. Program také obsahuje globální proměnnou `bool terminated`, kte-

rou nastavuje funkce, která je předána systému pro kontrolu, zda nedošlo k signalizaci ukončení. Pokud k ní dojde, je opět cyklus ve vláknu pro kontrolu vstupu ukončen a dále je ukončen i celý program.

3.3.3 freq

Program *freq* načítá ze standardního vstupu text (byty) a z těch vytvoří frekvenční tabulku. Hodnoty pro byty, které se objevují ve vstupu alespoň jednou vypíše do výstupu. Celý proces je implementován v cyklu `while`, který do bufferu o velikosti 1 ukládá vždy 1 načtený znak (byte). Pokud uživatel stiskne ENTER nebo CTRL + Z je načítání ukončeno a výsledek je vypsán na výstup. Frekvenční tabulka je vytvořena pomocí pole typu `int` o velikosti 256, což je počet ASCII znaků. Načtený znak je vždy převeden na číslo typu `int`, čímž je získán index do pole, na kterém se zvýší hodnota o 1. Data z tabulky jsou po ukončení vypsána v zadaném formátu.

3.3.4 echo

Echo vypisuje vše co je zadané v argumentu na výstup. V případě argumentu *"on"* nebo *"off"* zapne nebo vypne zobrazování aktuální pracovní složky v příkazové řádce. Implementována také byla možnost zobrazení nápovědy pomocí argumentu ve tvaru *"/?"*. V případě nezadání argumentů, dojde k vypsání nastavení, zda-li je echo zapnuté nebo vypnuté.

3.3.5 type

Program *type* slouží pro vypsání dat ze souboru nebo ze standardního vstupu. Po spuštění dojde ke kontrole, zda byly programu zadány nějaké argumenty. Pokud žádné argumenty zadány nebyly, program čte data ze standardního vstupu, handle je tedy standardní vstup. V případě, že argumenty byly zadány, jsou brány jako jeden celý řetězec, který reprezentuje název vstupního souboru a handle je pak handle souboru. Poté dochází k čtení dat do bufferu a jejich ukládání do c++ `stringu`. U konce dojde k vypsání celého `stringu`.

3.3.6 find

Find je program který je s argumenty */v /c""nazev_souboru* ekvivalentní programu *wc* v unixových systémech. Vypíše tedy počet řádek v souboru. Po spuštění je nutné zkontrolovat, zda jsou všechny parametry zadány správně. Poté dojde k otevření souboru a načtení jeho dat do proměnné typu `string`. Pokud není zadáno jméno souboru, je vstup čten buď z roury, nebo z konzole. Z této proměnné je poté vytvořen `stringstream`, ze kterého se postupně načítají řádky pomocí funkce `getline`. Počet řádek se poté vypíše.

3.3.7 dir

Pro vypsání všech položek ve složce slouží program *dir*. V případě, že nebyla zadána žádná složka v argumentu, jsou vypsány všechny položky z aktuální pracovní složky. Data ze složky jsou poté cyklicky načítána do bufferu typu `char`, tedy po bytech. Vždy je možné načíst maximálně 20 položek. Z bufferu jsou poté načítány jednotlivé položky které jsou převedeny na typ `TDir_Entry`. Následně jsou zjištěna potřebná data o souboru a ta jsou uložena do výstupního `stringu`, zároveň jsou ukládány informace o počtu složek a souborů. Na konci programu je výstupní `string` vypsán na výstup.

3.3.8 sort

Příkaz *sort* slouží pro seřazení načtených dat. Data jsou seřazena podle znaků v ASCII. Nejprve dochází ke zjištění, zda byly zadány argumenty. V případě, že žádný argument zadán nebyl, je jako vstup použit standardní vstup, který může být z konzole nebo roury. V opačném případě dochází k otevření souboru, ze kterého se budou data číst. Poté dochází ke čtení a k ukládání dat po řádkách do proměnné typu `string`. Po dokončení čtení dojde k uzavření souboru. Následně je proměnná typu `string` převedena na `istringstream`, pomocí kterého je vytvořen vektor řádků, nad kterým je poté použita metoda sort. Následně je proměnná typu `string` vyčištěna a jsou do ní načteny seřazené řádky. Výsledek je poté vypsán na výstup.

3.3.9 tasklist

Při použití programu *tasklist* dojde k vypsání probíhajících procesů. Informace o procesech jsou ukládány do souboru s názvem *"procfs"*. Tento soubor je poté otevřen a data z něj jsou následně vypsána na výstup. Poté dojde k zavření souboru.

3.3.10 cd

Program *cd* slouží pro změnu pracovního adresáře. Je implementován ve spouštěči příkazů, tedy `command_executor.cpp`. Nejprve dojde k zavolání funkce z knihovny RTL `Set_Working_Directory()`, který provede systémové volání pro změnu pracovního adresáře. Tato funkce bere jako parametr proměnnou typu `boolean`, do které se uloží, zda volání proběhlo správně. Pokud neproběhlo, dojde k vypsání chybového hlášení.

3.3.11 rd

Program *rd* se používá pro mazání položky souborového systému, tedy jak souboru, tak složky. Program zavolá funkci knihovny RTL, která provede systémové volání, které použije metodu ze souborového systému pro smazání položky.

3.3.12 md

Program *md* vytváří složku. Podobně jako příkaz rd pomocí funkce z RTL knihovny provede vytvoření složky se zadaným jménem.

3.3.13 shutdown

Příkaz *shutdown* zavolá z knihovny RTL funkci `Shutdown()` a ukončí simulaci operačního systému.

Kapitola 4

Závěr

Pro implementaci semestrální práce byla použita kostra, která byla dostupná na Courseware stránkách předmětu OS. Výsledná semestrální práce byla otestována pomocí přiloženého programu checker.exe.