Simulace operačního systému

KIV/OS - semestrální práce

tým:	Štěpán ŠEVČÍK	Jaroslav KLAUS	Radek VAIS
os číslo:	A17N0087P	A17N0074P	A17N0093P
zadání:			varianta 1
kontaktní email:			vaisr@students.zcu.cz
datum:			5.12.2017

1 Zadání

Vytvořte virtuální stroj, který bude simulovat OS, jehož součástí bude shell s gramatikou cmd. Vytvoříte ekvivalenty standardních příkazů a programů *echo*, *cd*, *dir*, *md*, *rd*, *type*, *wc*, *sort*, *ps*, *shutdown*, *rgen* a *freq*.

Na vybrané programy jsou kladené tyto speciální podmínky:

- **cd** musí akceptovat relativní cesty
- **type** musí umět vypsat jak stdin, tak musí umět vypsat soubor
- **rgen** bude vypisovat náhodně vygenerovaná čísla v plovoucí čárce na stdout, dokud mu nepřijde znak *Ctrl+Z* (*EOF*[1])
- **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 \n`“

Dále implementujte roury a přesměrování. Při simulaci nebudete přistupovat na souborový systém, ale vytvoříte prostředky simulátoru vlastní RAM-disk s názvem C.

Za 5 bonusových bodů můžete k realizaci souborového systému použít semestrální práci z KIV/ZOS - tj. implementace FAT.

[1] ang. End of File

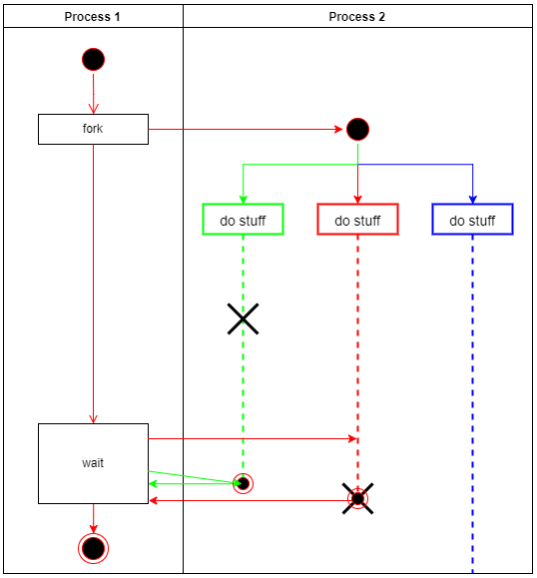
2 Analýza

2.1 Správa procesů

Správa procesů je modul OS, který udržuje informace o spuštěných procesech a jejich alokovaných prostředcích. Tyto informace jsou intenzivně používány modulem pro plánování procesů (není zadáním této práce), ale také I/O[2] modulem.

2.1.1 Životní cyklus procesu

V běžně používaných operačních systémech je definováno mnoho stavů procesu tak, aby OS mohl vyhodnotit, zda je možné naplánovat proces k vykonávání na procesoru. Tyto stavy popisují čekání na různé typy událostí, ze kterých lze odvodit připravenost procesu k naplánování. V této simulaci využijeme pouze tři základní stavy: běžící, čekající, ukončený. Během vytváření a plánování procesů může nastat několik situací souběhu událostí, které bude třeba ošetřit, viz Obrázek 1.



Obrázek 1: Diagram tří variant běhu processu potomka. Zelená ukazuje situaci, kdy užitečný kód potomka skončí před voláním `wait for`. Červená ukazuje ideální scénář, kdy proběhne volání `wait for` před koncem potomka. Modrá pak ukazuje vznik zombie procesu, který neskončil před koncem rodiče.

[2]modul vstupních a výstupníhc operací

Na základě Obrázku 1 bude třeba zajistit, aby obsluhu odstranění procesu spustil poslední proces, který přečte návratovou hodnotu. Zároveň by bylo vhodné vytvořt proces `init`, který by periodicky kontroloval zda v tabulce procesů není zombie proces, jehož rodič již neexistuje.

2.2 Vstupní a výstupní operace

Tento modul zapouzdřuje všechny vstupy a výstupy programů a jejich případné uložení. Zapouzdřuje tedy zápis do souboru nebo výpis na konzoli. Každé vstupní nebo výstuupní zažízení je identifikováno systémově unikátním označením (`handle`). Záznam do souboru bude pravděpodbně potřebovat jiný přístup než výpis na konzoli. Nabízí se vytvořit virtuální souborouvý systém, který pro různá zařízení použije vhodný ovladač. V rámci uživatelských programů tedy nebude rozdíl při čtení z konzole nebo ze souboru.

Použití virtuálního souborového systému umožní vytvořit libovolný ovladač například pro roury, standardní soubory nebo krátké záznamy, například seznam běžících procesů.

2.3 Shell

Shell je prvním uživatelským programem, který je spuštěn po spuštění jádra. V reálném OS by byl ještě před tím spuštěn program `login`, který by zajistil autentizaci uživatele.

Jednou z hlavních funkcí programu shell je interakce s uživatele. Pomocí něho je možné zadávat příkazy, spouštět další programy a číst jejich výsledky. Z toho vyplývá hlavní součást tohoto programu, tzv. parser. Parser přečte vstup z příkazové řádky a rozdělí jej na jednotlivé části. Z těchto částí poté setavuje příkazy a jejich argumenty, přiřazuje jim druh vstupních a výstupních proudů[3] a spouští je.

2.4 Uživatelské programy

Uživatelské programy jsou spouštěny programem shell použitím systémového volání pro spuštění nového procesu. Tím je zajištěno, že jim budou nastaveny správné vstupní a výstupní proudy a předány argumenty.

2.4.1 Program md

Program `md` slouží k vytvoření složek. Přijímá jeden argument, který určuje název složky k vytvoření.

[3] ang. stream

2.4.2 Program rd

Program `rd` slouží k mazání složek. Přijímá jeden argument, který určuje název složky k smazání. Dále je možné jako argument uvést `/S`, který má význam rekurzivního mazání a umožňuje tak vymazat neprázdný adresář. Uvedením argumentu `/Q` je možné aktivovat tichý režim, který se uživatele neptá, zda chce danou položku smazat.

2.4.3 Program dir

Program `dir` umožnuje uživateli zjistit obsah složky. Přijímá jediný parametr, a tím je cesta ke složce nebo souboru. Na jednotlivé řádky vypíše atributy souboru nebo složky, které jsou obsaženy v zadané složce, případně na jednu řádku vypíše atributy a název zadaného souboru.

2.4.4 Program echo

Program `echo` pouze vypisuje své argumenty. Do jedné řádky zřetězí všechny argumenty, které mu byly předány a oddělí je mezerami.

2.4.5 Program type

Program `type` slouží k vypsání obsahu souboru. Přijímá jediný parametr, který určuje soubor ke čtení.

2.4.6 Program sort

Program `sort` čte ze vstupu a řadí slova abecedně v sestupném pořadí. Slova v tomto pořadí poté vypíše. Pokud je zavolán s argumentem `/R`, vypíše slova v opačném, vzestupném pořadí.

2.4.7 Program wc

Program `wc` umožňuje spočítat slova na vstupu a tento počet poté vypíše.

2.4.8 Program ps

Program `ps` vypíše aktuálně běžící procesy v systému na výstup. Vypíše jejich PID a jejich název.

2.4.9 Program freq

Program `freq` provádí frekvenční analýzu bytů na vstupu. Pro každý byte, který byl zastoupen více než 0-krát, vypíše jeho četnost na výstup ve formátu „0x%hhx : %d“.

2.4.10 Program rgen

Program `rgen` vypisuje náhodná desetinná čísla v rozsahu 0-1 na výstup. Jeho činnost je ukončena, pokud na vstupu přečte znak `EOF`.

2.4.11 Program shutdown

Program `shutdown` ukončí všechny běžící procesy a ukončí činnost operačního systému.

3 Popis implementace

3.1 Správa procesů

Všechny prostředky správy procesů jsou implementovány ve složce `kernel/process`. Soubor `proces_api.h` poskytuje rozhraní služeb správy procesů pro jiné moduly jádra. Toto rozhraní deklaruje inicializační a ukončovací funkci správy procesů, vstupní bod obsluhy systémového volání služeb procesů, standardní rutiny pro práci s procesy (*getPID*, *getTID*, *getWD* ...) a převod indexování FD[4] z kontextu procesu na kontext jádra a zpět.

3.1.1 PCB a TCB

Modul proces spravuje dvě řídící tabulky: tabulku procesů a tabulku vláken. velikost těchto tabulek je řízena při překladu, omezuje tak maximální dostupné prostředky simulovaného OS. Každá z tabulek je složena ze záznamů kontrolních bloků PCB[5] nebo TCB[6].

PCB

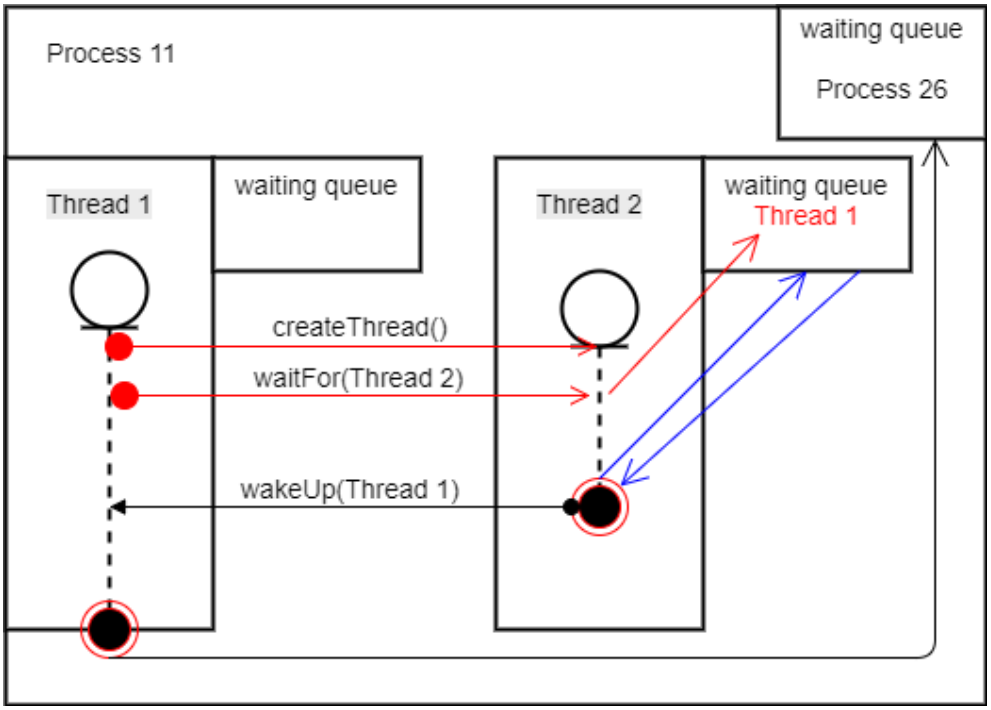
PID	Parent PID	Program name	Working directory	I/O devices list	Waiting queue	Result
THandle	THandle	std::string	std::string	std::vector<THandle>	waiting_queue	retval

Obrázek 2: Struktura kontrolního bloku procesu, včetně použitých datových typů

Význam položek PCB: `pid`, `parent pid`, `program name` a `working directory` viz Obrázek 2 je zřejmý. Seznam vstupních a výstupních zařízení (*io devices*) obsahue globální FD jádra k jednotlivým zařízením, která jsou uložena na indexu odpovídajícímu hodnotě FD pro process. Fronta čekajících (waiting queue) je vytovřená datová struktura obsahující seznam čekajících procesů/vláken na tento proces. Položka výsledek (result) je vytvořená datová struktura, která vypovídá o stavu procesu.

Datová struktura `process::waiting_queue` obsahuje seznam čekajících objektů (procesů nebo vláken) a přístup k ní je výhradní. Po skončení běhu procesu je třeba notifikovat všechny objekty v seznamu. Tato stuktura je nezbytná pro korektní naplnění podmínky deklarované v rozhraní `api.h`[7]. Zajišťuje tedy možnost vzbudit libovolný objekt viz Obrázek 3.

[4] z ang. *file descriptor*
[5] z ang. *process control table*
[6] z ang. thread control table
[7] operace čekej na hadle: „... funkce se vraci jakmile je signalizovan prvni handle ...“



Obrázek 3: Diagram vziku fronty čekajících a nastínění následné notifikace notifikace čekajících objektů.

Datová struktura `process::retval` uchovává návratovou hodnotu procesu a obsahuje čítač přečtení této hodnoty. Návratová hodnota je dostupná do doby, než ji přečtou všechny čekající procesy. Při přečtení návratové hodnoty posledním procesem jádro provede odstranění záznamu z tabulky procesů.

TCB

TID	PID	State	Waiting queue	Result
THandle	THandle	thread_state	waiting_queue	retval

Obrázek 4: Struktura kontrolního bloku vlákna, včetně použitých datových typů

Význam položek TCB: `tid`, `pid` viz Obrázek 4 je jasný. Součástí TCB je fronta čekajících a návratová hodnota. Tyto položky mají shodný význam jako u procesu. Navíc oproti procesu je zde položka stav (`thread state`), která umožňuje pozastavit proces v případě čekání na dokončení jiného. Pomocí teto podmínkové proměnné jsou vlákna notifikovány v případě, že je

voláno `Wake up`.

3.1.2 Životní cyklus procesu

Za základní plánovatelnou jednotku považujeme vlákno. Z toho vyplývá, že „nejmenší“ proces má právě jeden záznam v tabulce procesů i vláken. Algoritmus životnho procesu je následující:

1. Volání sysCall `Create_Process()`.
2. Vytvoření záznamu PCB v tabulce procesů.
3. Nastavení standardních streamů.
4. Příprava hlavního vlákna (vytvoření TCB záznamu).
5. Spuštění vlákna v crt0 spouštěného programu.
6. Užitečná práce spuštěného programu.
7. Nastavení návratové hodnoty.
8. Notifikace všech čekajích objektů.
9. Čekání na přečtení návratové hodnoty všemi čekajícími.
10. Vyčištění struktur TCB a PCB.

Úkolem funkce *crt0* uživatelských procesů je provést kopii argumentů a spustit výchozí funkci *main* daného programu se standardní deklarací argumentů int `main(int argc, char *argv[])`

Životní cyklus vlákna je prakticky shodný s životním cyklem procesu. Jediná změna je ve spuštění, kdy namísto *crt0* spouštíme uživatelem předanou referenci.

3.2 Virtuální souborový systém

Virtuální souborový systém neboli VFS [8] představuje abstrakční vrstvu mezi operačním systémem a konkrétními souborovými systémy. Díky VFS může OS a aplikace jej využívající používat jednotné rozhraní pro práci s například standardními proudy, rourami nebo s různými souborovými systémy.

[8] z ang. *virtual file system*

Do VFS je nejprve nutno zaregistrovat ovladač pro souborový systém, který poskytuje konkrétní implementace jednotlivých funkcí souborového systému, jako je například otevření souboru nebo zápis či čtení. Následně je možné do VFS navázat disk daného souborového systému pod značkou, jako je například `C`, `stdio` nebo `pipe`, skrze který aplikace přistupují k datům daného disku.

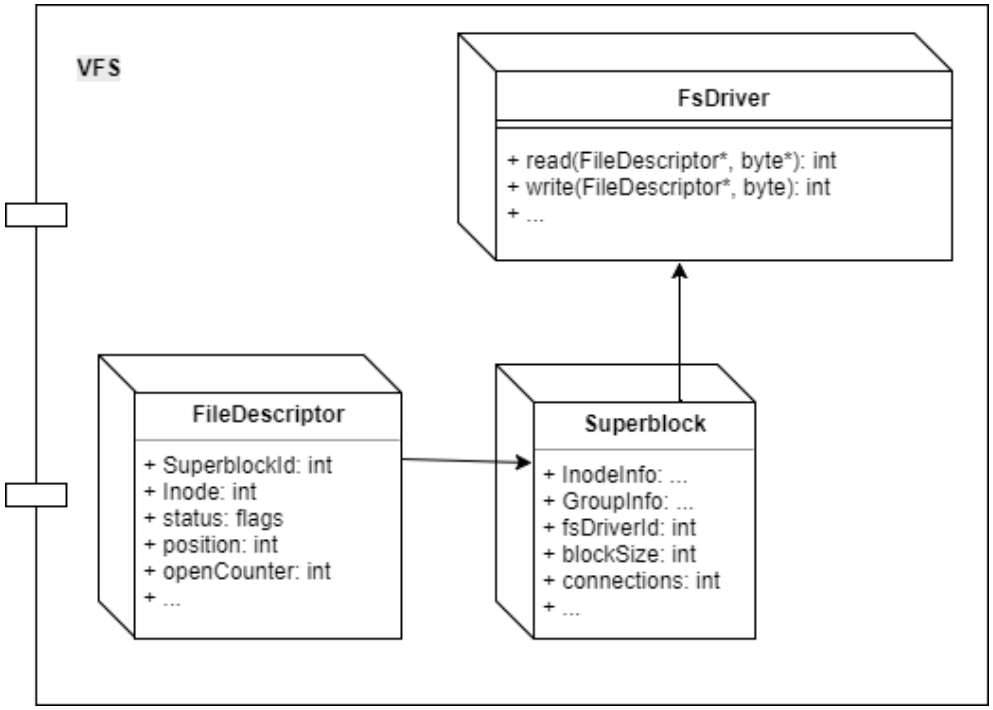
VFS se dále stará o normalizaci cesty a při každém přístupu, který využívá cesty, provádí následující transofrmace:

- zjistí, zda-li je cesta v absolutní podobě a pokud ne, prefixuje ji aktuálním pracovním adresářem právě běžícího procesu,
- sjednotí rozdělovače cesty, tj nahradí zpětná lomítka za dopředná lomítka,
- odstraní vícenásobné výskyty oddělovače cesty,
- odstraní každou část aktuálního adresáře, tedy jednu tečku,
- pro každý symbol „o adresář výše“, dvě tečky, odstraní část cesty,
- odstraní oddělovač cesty na konci cesty, pokud cesta není pouze značka disku.

3.2.1 Interní struktura VFS

Obrázek 5 znázorňuje vztahy mezi hlavními strukturami ve VFS. VFS udržuje v paměti zaregistrované ovladače souborových systémů v podobě struktur ukazatelů na funkce. Dále VFS udržuje záznamy o namountovaných discích v podobě takzvaných *Superblocků*, které obsahují identifikátor příslušného ovladače. Tyto struktury jsou v této simulaci statické a za běhu se kromě statistických údajů, jako například počítadla otevřených přístupů, nemění.

Ve VFS jsou dále file descriptory, které obsahují odkaz na disk, s jehož souborem pracují, a číslo inode souboru, se kterým manipulují. Dále každý file descriptor udržuje ukazatel do souboru na místo, od kterého bude prováděna příští čtecí či zapisovací operace anebo atributy souboru určující, zda-li se jedná o složku nebo zda-li je soubor otevřen pouze pro čtení. Tyto file descriptory jsou ve VFS ukládány v tabulce a jsou odkazovány pomocí univerzálního identifikátoru `handle`, který přímo odpovídá jejich pozici v tabulce. Exkluzivní přístup k file descriptorům mezi procesy je zajištěn mapováním procesových handlů na systémové handly.



Obrázek 5: Vztah mezi file dscriptory, superblocky a souborovými ovladači.

3.2.2 Vstupně/výstupní Operace

Po inicializaci VFS je nad ním možné volat následující vstupovýstupní funkce:

Otevřít soubor na zadané cestě. Hlavní parametr této metody je cesta, která je normalizována do absolutní podoby, která se skládá z názvu disku a z umístění souboru na disku. Tato funkce nejdříve nalezne příslušný disk a skrze jemu příslušící ovladač nechává otevřít konkrétní soubor.

Dalšími parametry jsou příznaky specifikující chování otevření a atributy souboru. Například příznak `fmAlways_Open` určuje, zdali má být soubor vytvořen či přepsán nebo pouze otevřen.

Mezi atributy potom patří například `faDirectory`, či `faSystem_File`, které jsou uloženy do file descriptoru.

Smazat soubor , obdobně jako otevřít soubor, volá na odpovídajícím ovladači pro smazání tohoto souboru.

Přečíst / zapsat operace pracují s file descriptory, pomocí kterých zjistí správný dísk, se kterým pracují a ovladač, který mají použít. Obě tyto ope-

race mají podobný průběh - další parametry po file descriptoru jsou odkaz na paměť, se kterou pracují, a počet bajtů ke čtení či zápisu. Po vyvolání odpovídající funckce na ovladači tyto operace vrací počet přečtených nebo zapsaných bajtů.

Získat / nastavit pozici souboru jsou operace, které také pracují s file descriptory. Obě sdílí parametr režim pozice, který může být jedním z následujících: **od začátku**, **od aktuální pozice** a **od konce**.

Operace nastavit pozici dále má parametr požadované pozice a příznak určující, zda-li má být požadovaná pozice také nastavena jako velikost souboru.

Tyto operace nemusí být poskytovány všemi ovladači protože například pro proudové souborové systémy nebo pro roury nemá pozice význam.

Zavřít soubor slouží pro zavření file descriptoru a uvolnění využitých prostředků. Některé souborové systémy mohou definovat speciální obsluhu pro uvolnění, jako například roury, které řeší synchronizaci produkujících a konzumujících vláken.

Například v případě zavření čtecí strany roury je třeba případnému čekajícímu vláknu signalizovat, že se nepovedlo zapsat všechna data.tecí strany roury je třeba případnému čekajícímu vláknu signalizovat, že se nepovedlo zapsat všechna data.

3.2.3 Souborové systémy

Pro potřeby této simulace byly vytvořeny následující souborové systémy:

FsStdio je obalující volání nad standardními proudy. Implementace čtení a zápisu jsou realizována pomocí knihovních primitiv jazyka C++.

FsPipe definuje roury, skrze které je možné přesměrovat proudy napříč řetězy programů.

FsProcess umožňuje číst informace o právě spuštěných procesech.

FsMemTree vychází z konceptu souborového systému ext2, ze kterého si propůjčuje ideologii inodů a bloků. Každý soubor a adresář je reprezentován právě jedním inodem, kterému je možné alokovat až N bloků.

3.3 Shell

Shell je prvním uživatelským programem, jehož úkolem je zpracovávat vstupy od uživatele, vyhodnocovat je, na jejich základě spouštět požadované procesy a vracet uživateli informaci a jejich průběhu nebo dokončení.

Shell v cyklu provádí několik úkonů. Jako první vypíše prompt - aktuální pracovní adresář. Poté pomocí systémového volání `Read_File` čte vstup, a ten nechá zpracovat parserem (podrobněji viz 3.3.1. Takto zpracovaný vstup předá do tzv. executora (podrobněji viz 3.3.2, který se stará o spouštění procesů.

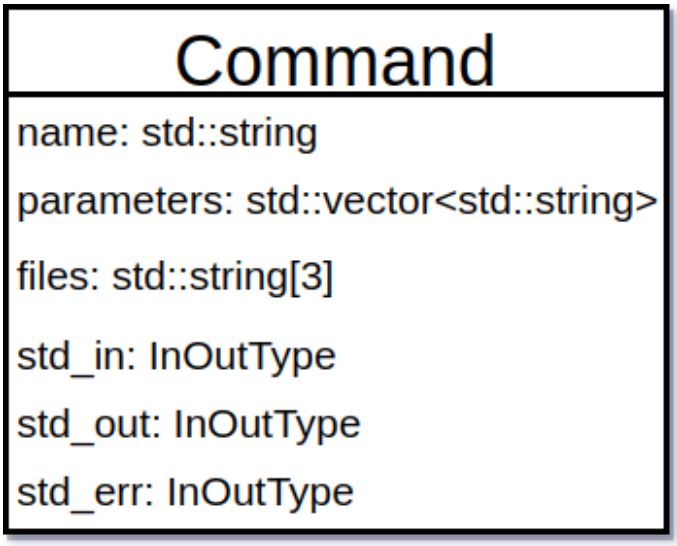
3.3.1 Parser

Shell obsahuje parser, který rozděluje načtený řetězec na symboly, které jsou nějakým způsobem jedním logickým celkem. Tyto symboly mohou být názvy programů, argumenty (jednoslovné i víceslovné v uvozovkách) a symboly určení vstupního nebo výstupního proudu. Tyto hodnoty poté seskupuje do struktury naznačené na Obrázku 6. Struktura obsahuje název programu, jeho parametry, pokud používá jako vstup nebo výstup soubory, tak názvy těchto souborů a typ vstupního, výstupního a chybového proudu. Tyto typy jsou definované hodnoty:

- STANDARD - dědí standardní proud od rodiče,
- PIPE - proud je jeden z konců roury[9],
- FILE_NEW - proudem je nový soubor,
- FILE_APPEND - proudem je soubor, na jehož konec se má zapisovat.

Pro přesměrování vstupu je možné použít znak '`<`' následovaný souborem, který se má použít jako vstupní proud. Další možností je mezi programy zapsat znak '`|`', který indikuje rouru (jeden konec slouží jako výstupní proud prvního programu, druhý jako vstupní proud druhého programu). Pro přesměrování výstupu je možné použí dříve zmíněnou rouru. Dále je také možné použít znak '`>`' následovaný názvem souboru. Ten je vytvořen (pokud existuje, je přepsán) a slouží jako výstupní proud. Použitím znaku '`>>`' následovaným názvem souboru se jako výstupní proud použije soubor, na jehož konec se bude zapisovat. Znak '`2>`' následovaný názvem souboru se přesměruje chybový výstupní proud do tohoto souboru.

[9] ang. pipe



Obrázek 6: Struktura příkazu vytvářená parserem.

3.3.2 Executor

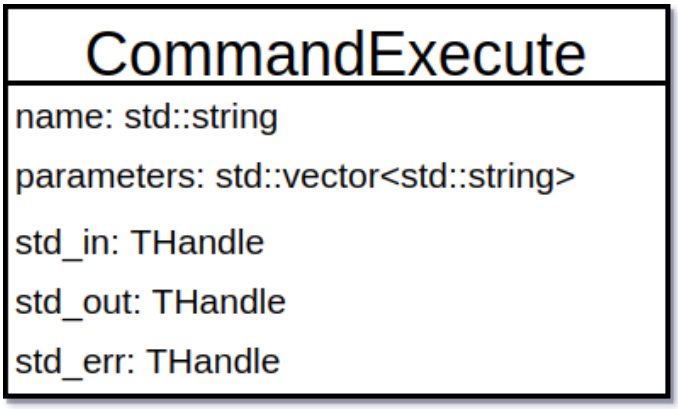
Další součástí shellu je tzv. executor, který se stará o spouštění programů. Nejprve zjistí, zda bude nový program dědit vstupní a výstupní proudy. Pokud ne, tak vytvoří podle potřeby buď rouru nebo otevře, případně vytvoří, soubor. Používá struktury podobné té na Obrázku 6. Ty již neobsahují jména souborů a proudy jsou jiného typu (file descriptor[10]), viz Obrázek 7. Poté systémovým voláním spustí pro každý program nový proces a zavře file descriptory předané procesu, aby zůstal správný počet otevření souborů. Nakonec se zablokuje čekáním na dokončení každého z programů.

3.4 Uživatelské programy

Protože uživatelské programy neměli obvyklou signaturu (`int main(int argc, char *argv[])`, zavedli jsme v naší simulaci konvenci, která tuto metodu v každém programu vytváří. Ve vstupním bodu programu získáme argumenty a předáme je této funkci. Získání argumentů je implementováno v `common.cpp` a je jakousi obdobou `crt0`.

Uživatelské programy v implementaci používají typicky standardní knihovny jazyka `C++11`, jako je `std::string`, `std::vector`, `std::map` nebo `std::stringstream`. K provádění operací, které jsou v uživatelském kontextu zakázané, používá knihovnu `kiv_os_rtl`, která poté provádí systémová volání. Přepnutí kon-

[10] zkráceně fd, jinak také handle



Obrázek 7: Struktura příkazu vytvářená executorem.

textu je simulováno předáním struktury `TRegisters`, která je jako jediná předávána mezi uživatelským režimem a režimem jádra.

4 Uživatelská příručka

4.1 Překlad

Pro překlad zdrojových souborů je vhodné použít MS Visual Studio verze 2017. Ve složce `msvc` je připraveno řešení (solution) v souboru *OSSimulator.sln*, které sjednocuje jednotlivé projekty `boot`, `kernel` a `user`. Každý projekt pak obsahuje soubory, jejichž závislosti je třeba vyřešit při manuálním překladu. Pro jednotlivé projekty můžeme volit typ sestavení (*debug* / *release*).

4.2 Spuštění

Ke spuštění programu je možné použít nástroje MSVS[11], nebo spustit program `boot.exe`, který při použití projektu MSVS bude sestaven do složky `compiled`. Je možné přesměrovat vstup programu a simulovat tak zadávané příkazy sekvencí v souboru.

Po spuštění je zaveden prázdný RAM-disk s názvem C: a spuštěn shell v C:/. Program nyní čeká na příkazy shellu viz kapitola Ovládání.

[11]Microsoft Visual Studio

4.3 Ovládání

Po spuštění se zobrazí příkazová řádka, která čeká na vstup od uživatele.Takovým vstupem může být buď volání jednoho programu, nebo sekvence programů, jimž jsou logicky správně přiřazeny vstupní a výstupní proudy. Příklady takového vstupu jsou např.:

- `md folder`
- `echo Some text > file.txt`
- `echo Some text | wc`
- `type file | sort | wc`

Nastavení vstupních a výstupních proudů se provádí následujícími symboly:

- '`<`' následovaný názvem souboru nastavuje vstupní proud na daný soubor
- '`|`' indikuje rouru, jejíž jeden konec slouží jako výstupní proud prvního programu a druhý jako vstupní proud druhého programu
- '`>`' následovaný názvem souboru nastaví výstupní proud na nový soubor (přepíše existující)
- '`>>`' následovaným názvem souboru se jako výstupní proud použije soubor, na jehož konec se bude zapisovat
- '`2>`' následovaný názvem souboru se přesměruje chybový výstupní proud do tohoto souboru

4.3.1 Program md

Program `md` slouží k vytvoření složek. Přijímá slespoň jeden argument, který určuje název složky k vytvoření.

4.3.2 Program rd

Program `rd` slouží k mazání složek (i souborů). Přijímá alespoň jeden argument, který určuje název složky k smazání. Dále je možné jako argument uvést `/S`, který má význam rekurzivního mazání a umožňuje tak vymazat neprázdný adresář. Uvedením argumentu `/Q` je možné aktivovat tichý režim, který se uživatele neptá, zda chce danou položku smazat.

4.3.3 Program dir

Program `dir` umožnuje uživateli zjistit obsah složky. Přijímá jediný parametr, a tím je cesta ke složce. Na jednotlivé řádky vypíše atributy souborů nebo složek a jejich název, které jsou obsaženy v zadané složce. Pokud parametr není zadán, uvažuje se aktuální pracovní adresář.

4.3.4 Program echo

Program `echo` pouze vypisuje své argumenty. Do jedné řádky zřetězí všechny argumenty, které mu byly předány, a oddělí je mezerami.

4.3.5 Program type

Program `type` slouží k vypsání obsahu souboru. Přijímá alespoň jeden parametr, který určuje soubor ke čtení.

4.3.6 Program sort

Program `sort` čte ze vstupu a řadí slova abecedně v sestupném pořadí. Slova v tomto pořadí poté vypíše. Pokud je zavolán s argumentem `/R`, vypíše slova v opačném, vzestupném pořadí.

4.3.7 Program wc

Program `wc` umožňuje spočítat slova na vstupu a tento počet poté vypíše.

4.3.8 Program ps

Program `ps` vypíše aktuálně běžící procesy v systému na výstup. Pokud je zavolán s přepínačem `-pid`, vypíše i jejich PID.

4.3.9 Program freq

Program `freq` provádí frekvenční analýzu bytů na vstupu. Pro každý byte, který byl zastoupen více než 0-krát, vypíše jeho četnost na výstup ve formátu ”0x%hhx : %d”.

4.3.10 Program rgen

Program `rgen` vypisuje náhodná desetinná čísla v rozsahu 0-1 na výstup. Jeho činnost je ukončena, pokud na vstupu přečte znak `EOF`. Program se skládá ze dvou vláken - jedno generuje a vypisuje čísla a druhé čte vstupní proud a čeká na `EOF`.

4.3.11 Program shutdown

Program `shutdown` incializuje ukončovací sekvenci operačního systému, která zahrnuje ukončení všech programů a následné ukončení jádra.

5 Závěr

V rámci této práce jsme vytvořili knihovnu `kernel.dll`, která simuluje jádro operačního systému. Jádro obsahuje implementaci základní správy procesů a vláken (vytvoření, čekání). Pro I/O operace jsme implementovali virtuální souborový systém s ovladači pro *RAM-disk*, *roury*, *stdio* a *procfs*.

Dále jsme vytvořili knihovnu `user.dll`, která představuje sadu uživatelksých programů a knihoven dle zadání. Knihovna `user` *rtl* využívá rozhraní `SysCall` knihovny `kernel` k vykonávání systémových volání.

Standard deklarovaný souborem `api.h` jsme rozšířili o jeden chybový kód a dvě systémová volání.

- `erProces_Not_Created` - návratový kód chyby vytváření procesu
- `scGetFileAttributes` - systémové volání pro zjištění atributů souboru
- `scShutdown` - systémové volání pro modul proces, který inicializuje vypnutí jádra OS