FAKULTA
APLIKOVANÝCH VĚD
ZÁPADOČESKÉ
UNIVERZITY
V PLZNI

KIV/OS - Semestrální práce

Simulátor operačního systému

student: Daniel HRYZBIL, Anežka JÁCHYMOVÁ, Zdeněk VALEŠ
datum: 29.11.2019

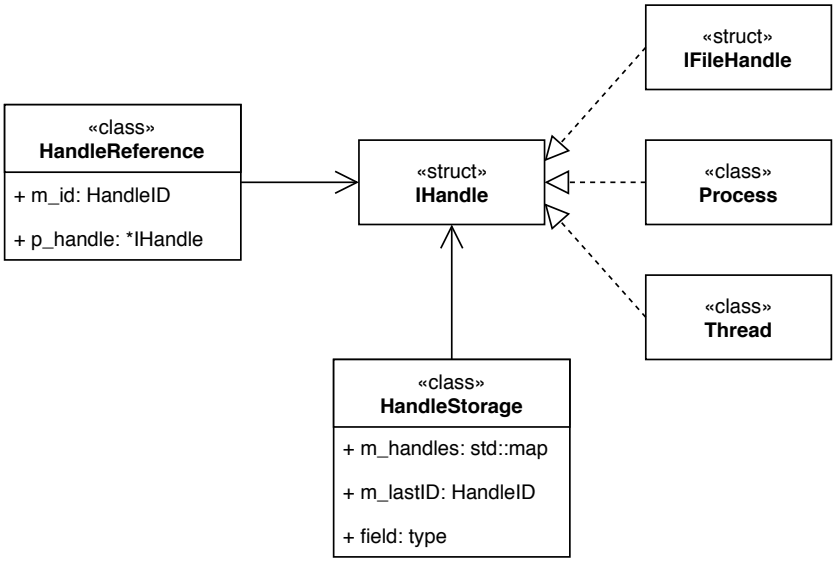
1 Zadání

1. Vytvořte virtuální stroj, který bude simulovat OS
2. Součástí bude shell s gramatikou cmd, tj. včetně exit
3. Vytvoříte ekvivalenty standardních příkazů a programů:
   a) echo, cd, dir, md, rd, type, find /v /c""(tj. co dělá wc v unix-like prostředí), sort, tasklist, shutdown
      i. cd musí umět relativní cesty
      ii. echo musí umět @echo on a off
      iii. type musí umět vypsat jak stdin, tak musí umět vypsat soubor
   b) Dále vytvoříte programy rgen a freq
   c) rgen bude vypisovat náhodně vygenerovaná čísla v plovoucí čárce na stdout, dokud mu nepřijde znak Ctrl+Z //EOF
   d) 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"
4. Implementujte roury a přesměrování
5. Nebudete přistupovat na souborový systém, ale použijete simulovaný disk
   a) Za 5 bonusových bodů můžete k realizaci souborového systému použít semestrální práci z KIV/ZOS - tj. implementace FAT.

2 Kernel

Jádro poskytuje API pro správu procesů a vláken, vytvoření pipe a přístup k disku. TODO: trochu rozvést

Základním konceptem kernelu je třída `HandleStorage` společně s `HandleReference`. Každá třída reprezentující nějaký handle (soubor, proces, vlákno, ...) implementuje rozhraní `IHandle`. Všechny instance těchto tříd jsou následně uloženy uvnitř `HandleStorage`, kde jim je přiřazeno jejich unikátní `HandleID` (16 bitové číslo). Přístup k uloženým handle je následně možný pouze přes objekty `HandleReference`. Struktura handlerů je znázorněna diagramem tříd na obrázku 1.



Obrázek 1: Diagram tříd handlerů

Uvnitř `HandleStorage` je pro každý handle uložený počet referencí na tento handle, který se inkrementuje vždy při vytvoření nové `HandleReference` a dekrementuje při odstranění `HandleReference`. Handly s nulovým počtem referencí jsou automaticky uzavřeny a odstraněny ze systému.

Každý proces si udržuje vlastní množinu referencí na handly, které byly v jeho kontextu vytvořeny nebo jinak získány (například při vytváření procesu předá rodičovský proces nově vytvořenému procesu handle na `stdin` a `stdout`). Tato množina se zároveň používá i pro zjištění, zda má proces k nějakému handlu vůbec přístup. Při odstranění nějakého procesu potom dojde k odstranění všech handle referencí uložených v jeho množině, a tím dojde automaticky i k odstranění všech handlů, které používal pouze daný proces.

Kernel se tak nespoléhá na "slušnost"user-space kódu, který by měl vždy použít `CloseHandle`, ale dokáže při ukončení procesu automaticky uklidit všechny nepotřebné handly, stejně jako reálný OS. Zároveň tento koncept umožňuje i efektivnější spolupráci více vláken najednou. Při práci s nějakým handle totiž není potřeba zamykat globální zámky, protože pokud pro tento handle existuje alespoň jeden objekt `HandleReference`, tak je zaručeno, že žádné jiné vlákno nemůže tento handle nečekaně odstranit a způsobit tak pád celého systému.

2.1 Procesy a vlákna

Procesy jsou uloženy v již zmíněném `HandleStorage` a `HandleID` je bráno jako PID. Stejně tak jsou uložena i vlákna a `HandleID` představuje TID.

Každé vlákno simulovaného OS využívá reálné vlákno fyzického OS (`std::thread`). Kernel dále neobsahuje žádný platform-specific kód, kromě načítání "user.dll". Za hlavní vlákno procesu se považuje jeho první vlákno. Při vytvoření procesu je vždy vytvořeno

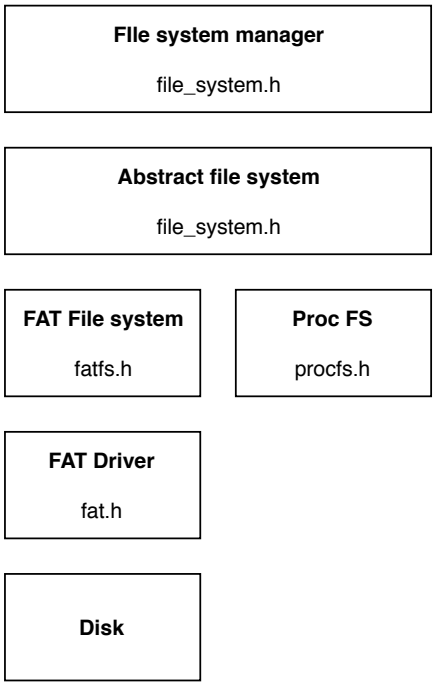
i jeho hlavní vlákno.

2.1.1 API

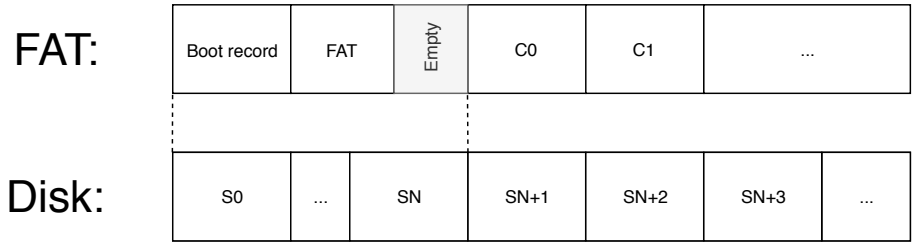
- CreateProcess
- Clone
- CreateThread
- WaitFor
- GetExitCode
- Exit
- SetupSignal
- SystemShutdown

2.2 Filesystem

Kernel obsahuje file system manager, který spravuje disky a souborové systémy na nich uložené. Pro přístup na disk je použito abstraktní API, které každý ovladač pro FS. Struktura je znázorněna na obrázku 2. Jako filesystem jsme zvolili FAT implementovanou v rámci KIV/ZOS.



Obrázek 2: Struktura správy souborového systému



Obrázek 3: Uložení FAT souborového systému na disk

2.2.1 ProcFS

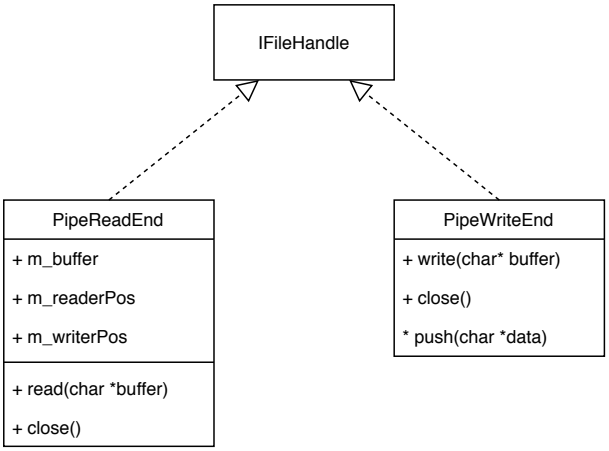
Na virtuálním disku 0. Používá se k získání právě běžících procesů.

2.2.2 API

- Create
- Query
- Read
- ReadDir
- Remove
- Resize
- Write

2.3 Pipe

Pipe je vytvořena dvěma handly - `PipeReadEnd` a `PipeWriteEnd`. Viz obrázek 4.



Obrázek 4: Pipe diagram tříd

3 RTL

4 Shell a uživatelské příkazy

5 Závěr