Semestrální práce z KIV/OS

Simulace operačního systému

Marek Šimůnek A15N0082P

Jindřich Pouba A15N0072P

Matěj Lochman A15N0068P

27. listopadu 2015

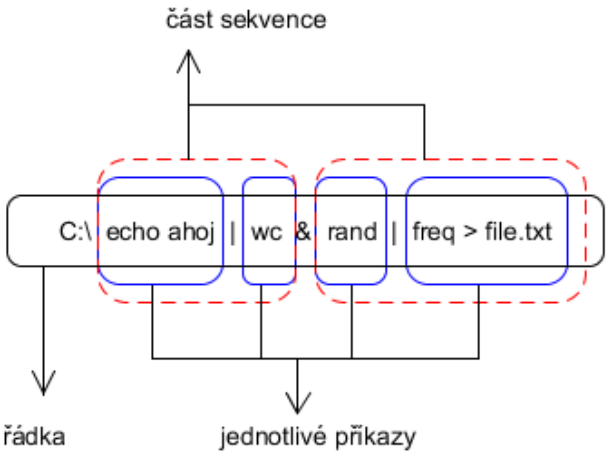
1 Zadání

- Vytvořte virtuální stroj, který bude simulovat OS
- Součástí bude shell s gramatikou cmd
- Vytvoříte ekvivalenty standardních příkazů a programů
  - echo, cd, dir, md, rd, type, wc, sort
  - Dále vytvoříte programy rand a freq
  - rand 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 uděláte si prostředky simulátoru vlastní RAM-disk s názvem C

2 Implementace

2.1 Souborový systém

Souborový systém je organizován do stromové struktury (v `node.h`). Data jsou ukládána do souvislého bloku paměti. K souboru může přistupovat pouze jeden proces. Složka se od souboru rozlišuje příznakem `directory`.



Obrázek 1: Parsování příkazů

2.2 Parser

Zpracování vstupu se provádí v souboru `c_cmd.cpp`. Řádka je nejprve rozdělena na sekvenční části, které odděluje znak `;`. Sekveční části se pak parsují podle rour na jednotlivé příkazy. Ze samotných příkazů se pak získá název příkazu, parametry a přesměrování.

2.3 Roura

Roura (`pipe.h`) je implementována kruhovým bufferem. Jako synchronizační mechnanismus jsou použity dva semafory a jedna kritická sekce. Má dva handlery - jeden pro čtení, druhý pro zápis.

2.4 Seznam příkazů

Seznam implementovaných příkazů, které můžete zadávat do příkazové řádky:

- cmd, echo, cd, tree
- dir, mkdir (md), rm (rd)
- freq, rand, type
- scan, který vypisuje ascii hodnotu ze vstupu
- pipe, který vypisuje vstup na výstup
- info, který vypíše parametry
- exit ukončí cmd proces

  Lze zadávat sekvence příkazů oddělené `;` v rámci jednoho vstupního řádku. Příkazy lze také přesměrovávat a spojovat.

2.5 Spouštění příkazů

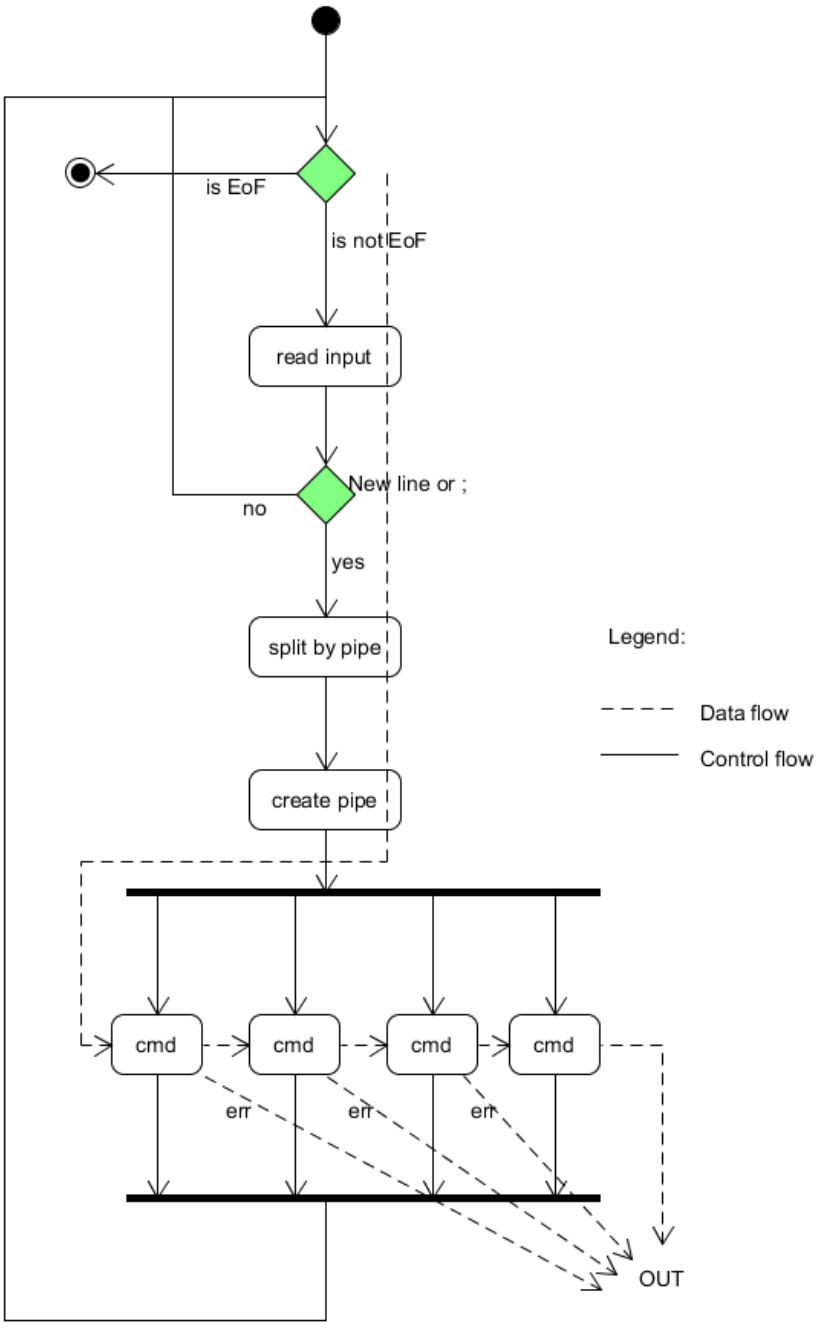
Hlavní proces cmd čte vstup, který parsuje. Pokud narazí na středník nebo novou řádku přestane parsovat vstup a zpracuje příkaz. Příkaz se parsuje podle rour. Za každý znak roury se vytvoří nová roura. První příkaz pak dostane vstup od cmd a poslední příkaz předá výstup a chybové hlašení zpátky cmd. Následně se příkazy spojí pomocí rour. Po spojení se všechny příkazy spustí a čeká se na poslední příkaz. Až doběhne předá se řízení cmd, který čte opět vstup.

3 Uživatelská příručka

Pro spuštění programu stačí otevřít spustitelný soubor *os.exe*. V terminálovém okně se zobrazí aktuální adresář, ve kterém se nacházíte. V tuto chvíli můžete zadávat dostupné příkazy (viz 2.4).

Každý příkaz může být přesměrován na výstup zápisem `příkaz > soubor` a na vstup `příkaz< soubor`. Lze i přesměrovat na konec souboru konstrukcí >>. Příkazy je dále možné spojovat pomocí rour pro nasměrování výstupu jednoho procesu na vstup následujícího procesu `příkaz1 | příkaz2`.

Ukončení čtení se provádí CTRL+Z. V spuštěné příkazové řádce lze spustit další příkazovou řádku. Každá příkazová řádka se ukončuje příkazem `exit`.



Obrázek 2: Data a control flow diagram

4 Závěr

V rámci této práce jsme implementovali zadané příkazy, roury a přesměrování a splnili jsme tak všechny body zadání. Snažili jsme se vyvarovat zmíněným

častým chybám a dbali jsme na doporučení uvedená na CW - spuštění shellu z shellu atp. Nad rámec zadání jsme umožnili přesměrování chybového výstupu a také jsme implementovali příkaz scan, který vypisuje ASCII hodnoty znaků ze vstupu na výstup.

Aplikaci jsme během vývoje průběžně testovali, a díky tomu se nám podařilo odstranit velké množství chyb. U většiny příkazů jsme neimplementovali jejich parametry, ale v rámci rozšíření práce by bylo možné tuto funkcionalitu dodělat. Díky této semestrální práci jsme si osvěžili práci s pamětí, ukazateli a vlákny, avšak hlavním přínosem bylo hlubší pochopení toho, jak vlastně shell funguje.