ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

KIV/OS

Simulace operačního systému v jazyce Java

Řešitelé: BLÁHA Martin, A14N0119P
KOŠEK David, A14N0132P
NEUMANN Antonín, A14N0139P
E-mailová adresa: kivos@post.cz
Akademický rok: 2014/2015

1 Zadání

Úkolem semestrální práce bylo vytvořit program v jazyce Java simulace operačního systému. První částí práce bylo navrhnout a implementovat strukturu procesů a jejich spouštění, vzájemnou komunikaci mezi procesy, I/O operace, atp.

Druhou částí byla implementace základní množiny příkazů (programů) pro navrženou strukturu operačního systému.

Povinné příkazy shellu

- cat - vypíše soubor
- cd - změní pracovní adresář shellu
- echo - vypíše zprávu
- exit - ukončí shell
- kill - ukončí proces
- ls - vypíše obsah pracovního adresáře; ls - vypisuje aktuální adresář; ls rel_cesta; ls abs_cesta
- man - vypíše stručnou nápovědu a všechny implementované příkazy
- ps - vypíše všechny procesy v modelu (výpis kompletně všech procesů - kvůli ladění)
- pwd - vypíše pracovní adresář shellu
- shell - spustí shell (lze spouštět rekurzivě)
- shutdown - ukončí běh modelu; ne pouhé volání Sytem.exit, ale volání služby, která nejprve ukončí všechny procesy, ... a až na konec zavolá System.exit (pouze pokud je potřeba)
- -sort - seřadí řádky ze vstupu a vypíše je na výstup

Každý z uvedených příkazů musí obsahovat jednotnou možnost získání nápovědy o funkci, možných parametrech atd. Dále je nutné uchovávat historii příkazů.

2 Uživatelská dokumentace

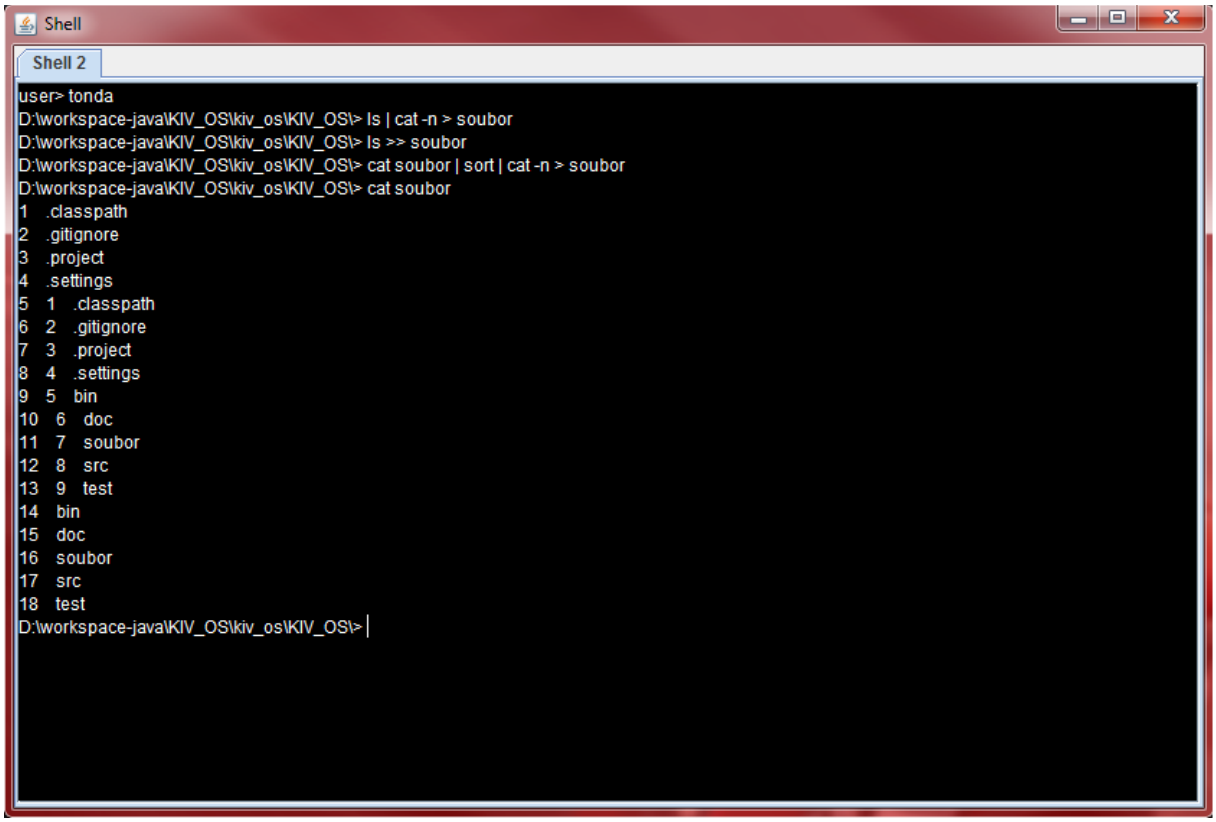
Po spuštění aplikace je potřeba zadat uživatelské jméno (login), toto jméno je zcela libovolné. Heslo není potřeba zadávat.

Všechny dostupné příkazy se vypíší po zadání příkazu `man` bez parametru. Příkazem `man` následovaným názvem příkazu získaným způsobem popsaným výše, je možné získat nápovědu, správnou syntaxi a přehled povolených parametrů tohoto programu.

Kromě volání příkazů a jejich parametrů je možné každému příkazu přesměrovat standardní vstup ze souboru, respektive výstup do souboru. Taktéž je možné více příkazů „zřetězit“ za použití tzv. rour (anglicky pipe), čímž docílíme přesměrování výstupu jednoho příkazu na vstup druhého.

Ukázka příkazů, jejich přesměrování a řetězení

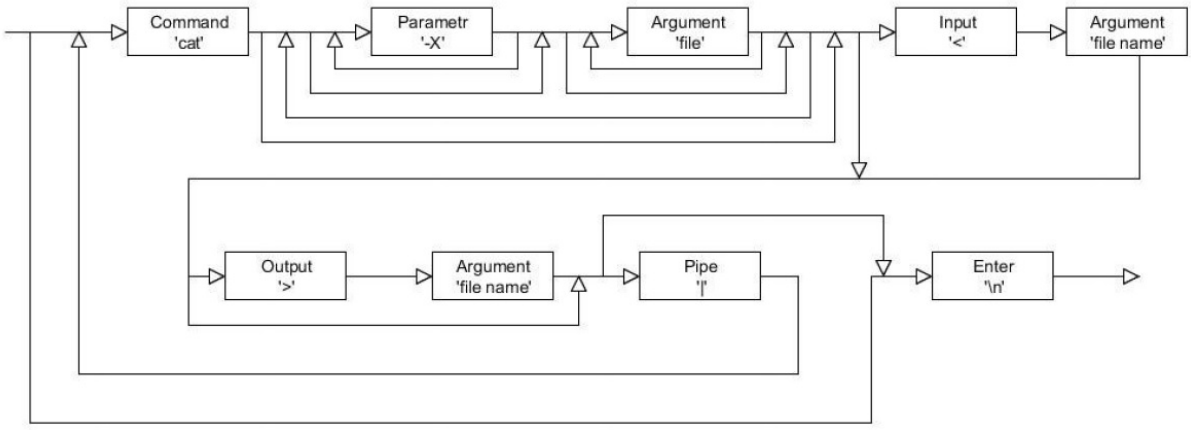
```
ls | cat -n > soubor
ls >> soubor
cat soubor | sort | cat -n > soubor
cat soubor
```



3 Programátorská dokumentace

Gramatika

Veškeré vstupy, co jsou zadané do terminálového okna jsou předány do syntaktického analyzátoru. Ten provede potřebné úpravy se vstupem a rozhodne, zda-li vstup byl zadán správně, či nikoliv. Analyzátor si nejprve ze vstupu odstraní přebytečné mezery, poté následuje analýza jednotlivých částí. Důležité je kontrolovat, aby vstup ze souboru byl před výstupem do souboru, samozřejmě při použití roury lze vše opakovat. K tomu to využívá analyzátor nastavování jednotlivých příznaků. Další důležitou věcí, co kontrolujeme je, aby za vstup/výstupem byl argument (očekává se soubor). Pro představu, co analyzátor akceptuje můžeme vidět na obrázku umístěný pod textem.



Pokud vstup odpovídá námi navržené gramatice dochází k nastavení jednotlivých parametrů do příslušných proměnných. K tomu nám pomáhá objekt, který má právě tyto proměnné, který poté ukládáme do listu typu těchto objektů. O toto nastavení se stará sémantický analyzátor. S tímto listem poté pracujeme dál a voláme jednotlivé programy.

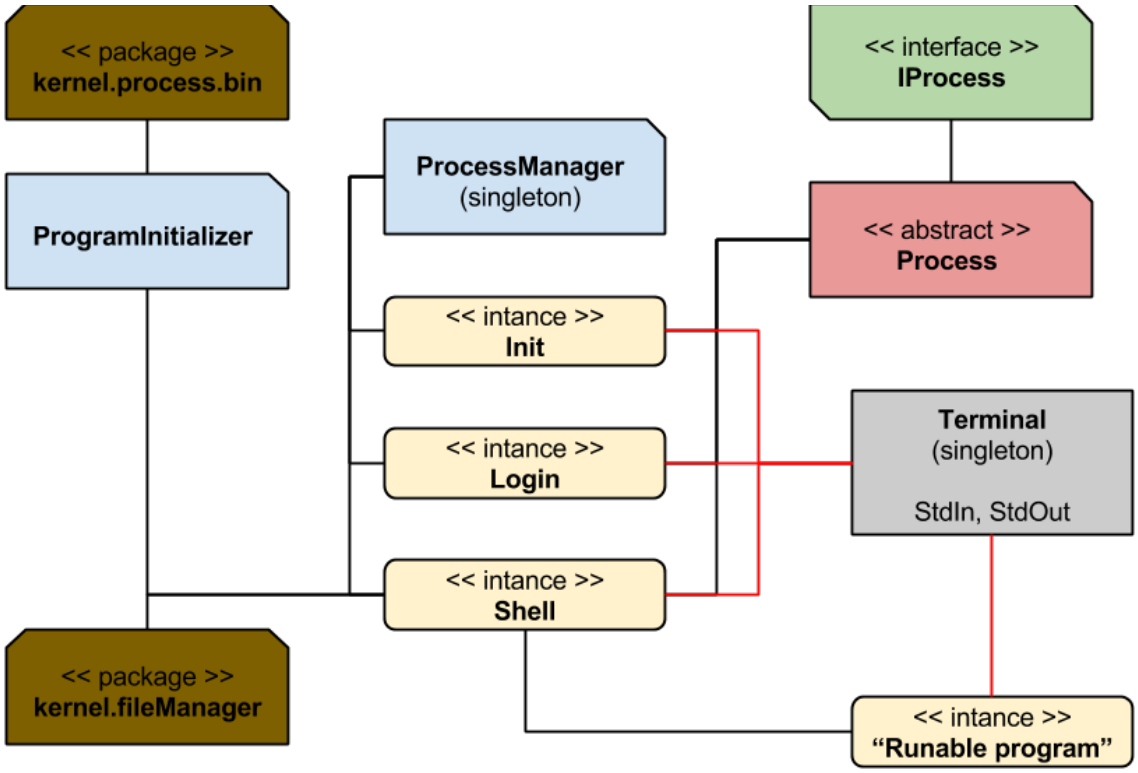
Základní kostra aplikace

Základním prvky našeho modelu operačního systému jsou abstraktní třída Process, ProcessManager, ProgramInitializer a procesy Init a Shell.

Třída Process definuje proměnné a metody přístupné pro každý implementovaný příkaz. Třídy ProccessManager a ProgramInitializer se starají o správné spuštění jednotlivých příkazů a nastavení správných hodnot proměnných jako například vstupní a výstupní proudy.

Instance třídy Init je prvním procesem, který se po spuštění aplikace vytvoří a má za úkol spustit přihlášení uživatele a následně vytvořit první instanci procesu Shell.

Shell je proces, který běží po celou dobu běhu aplikace, tzn. že s ukončením poslední běžící instance Shellu se ukončí i celá aplikace. Každá jeho instance si vytváří samostatné terminálové okno a je schopná vykonávat příkazy nezávisle na ostatních instancích shellu.



Ilustrace 1: Model základní architektury aplikace

Spouštění procesů, vstupy, výstupy, roury

Pro spuštění procesů pomocí příkazů se stará proces shell. Zadaný příkaz se nejdříve ověří pomocí syntaktického analyzátoru, jestli je zadaný příkaz v pořádku a následně ho pomocí sémantického analyzátoru zpracuje. Spuštění procesu probíhá tak, že pomocí třídy ProgramInitializer se vytvoří instance procesu, které se následně nastaví vstupní a výstupní proud. Ze vstupního proudu může proces načítat data, která se načítají ze souboru, nebo z terminálu, nebo je zapisuje předchozí proces. Proces poté může zapisovat do výstupního proudu, který zapisuje do souboru, do nadcházejícího procesu, nebo na terminál. Poté se procesu nastaví vstupní parametry. Nakonec je proces předán proces manageru, který daný proces spustí a následně se čeká na jeho dokončení.

Souborový manažer

Souborový manažer, jak už název napovídá slouží pro práci s adresáři a soubory. V manažeru lze soubory či adresáře vytvářet a mazat. Samozřejmostí manažeru je zápis a čtení ze souboru. Další funkcionalita souvisí s načítáním adresářové struktury. Pomocí příkazu ls, který vypíše na standardní výstup adresáře a soubory. Tento výpis lze specifikovat pomocí parametrů (například zobrazení skrytých souborů), které manažer akceptuje.

4 Závěr

Splnění zadání

Zadání jsme splnili ve všech bodech. Kromě implementace povinných příkazů, jsme navíc implementovali příkazy `del` (smaže zvolený soubor nebo adresář), `mv` (přesune/přejmenuje zvolený soubor nebo adresář), `cp` (překopíruje zvolený soubor nebo adresář) a `mkdir` (vytvoří nový adresář). Tyto příkazy jsme implementovali zejména kvůli pohodlnější práci v terminálu aplikaci, aby nebylo neustále nutné přepínat se z aplikace do operační systému. Dále jsme práci rozšířili na víceúlohový systém, pomocí příkazu `shell` se nám otevře nová záložka (nové terminálové okno) a fungují obě spuštěné okna. V terminálovém okně jsme pro lepší testování rozšířili práci i o to, aby prompt ukazoval celou cestu k aktuálnímu adresáři.

Práce jednotlivých členů týmu

Každý člen týmu se aktivně podílel na vývoji aplikace a to během celého období. David Košek pracoval zejména na gramatice, syntaktickém analyzátoru a správci vstupně výstupních operací. Antonín Neumann vypracoval a implementoval základní strukturu aplikace a implementoval většinu příkazů. Martin Bláha rozvedl implementaci struktury aplikace, navrhl a implementoval terminálové okno, příkaz shell a práci ve vstupy, výstupy a rourami.

Spolupráce týmu byla ovšem mnohem užší a celý tým těžil ze znalostí každého jeho člena. Testování a opravu chyb dělali všichni členové týmu.

Subjektivně-objektivní zhodnocení aplikace

Mezi silné stránky naší aplikace bychom rozhodně zařadili systém volání jednotlivých příkazů, který jsme se snažili co nejvíce zjednodušit, aby bylo přidání jednotlivých příkazů co nejjednodušší a nezáviselo na znalosti architektury modelu operačního systému. K vytvoření nového příkazu je potřeba pouze vytvořit příslušnou třídu v balíku kernel.process.bin se stejným názvem jakým se později bude příkaz volat v shellu pouze s velkém písmenem na začátku (podle konvencí Java). Dále je potřeba ve vytvořené třídě dědit od abstraktní třídy Process a přepsat metodu run(), která je obdobou metody main(), jež musí obsahovat každý spustitelný program.

Mezi slabé stránky naší aplikace bychom mohli zařadit nedostatečné otestování celé simulace operačního systému z důvodu nedostatku času a celkového vytížení týmu. Práce by se dala rozšířit o automatické testy.

Závěrečné zhodnocení

Práce nám ukázala rozdílnost v myšlení i ve způsobech programování každého z nás a přinutila nás ke spolupráci, která určitě nebyla vždy snadná. Rovněž jsme se zdokonalili v používání mechanizmů určených pro spolupráci v týmu, zejména nástroje GIT.