FAKULTA
APLIKOVANÝCH VĚD
ZÁPADOČESKÉ
UNIVERZITY
V PLZNI

OSMRR - Simulace operačního systému

Semestrální práce KIV/OS

E-mail:

jmoulis@students.zcu.cz

Řešitelé (abecedně):

Jan Moulis, A13N0118P

Kamil Rendl, A13N0123P

Jakub Rinkes, A13N0124P

5. 12. 2013

1 Zadání

Navrhněte a implementujte model operačního systému. Jako součást operačního systému vytvořte interpret shell, vycházející z linuxové obdoby bash.

1.1 Příkazy shellu

- CAT vypíše soubor
- CD změní pracovní adresář shellu
- ECHO vypíše zprávu
- EXIT ukončí shell
- KILL ukončí proces
- LS vypíše obsah pracovního adresáře
- MAN vypíše stručnou nápovědu (implementováno jako příkaz help)
- PS vypíše všechny procesy v modelu
- PWD vypíše pracovní adresář shellu
- SHELL spustí shell
- SHUTDOWN ukončí běh systému
- SORT seřadí řádky ze vstupu a vypíše je na výstup

2 Uživatelský manuál

2.1 Minimální systémové požadavky

- Windows XP a novější
- Java 1.5.0 nebo novější

2.2 Sestavení programu

Přeložení programu se provede spuštěním dávkového souboru *compile.bat*, který přeloží zdrojové soubory do složky *build*.

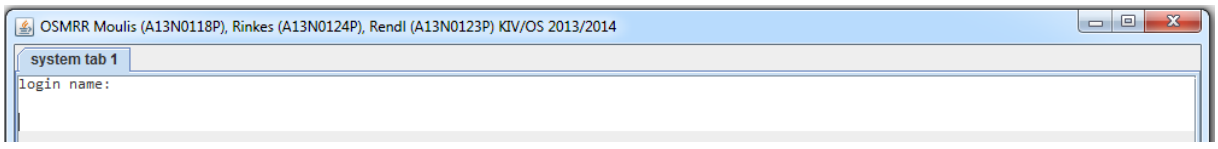
Poznámka:

Složka help musí být v době překladu umístěna na úrovni dávkového souboru compile.bat. V opačném případě nebudou dostupné nápovědy příkazů, ale systém bude plně funkční.

2.3 Spuštění

Pro spuštění programu slouží dávkový soubor *run.bat*.

Na obrazovce se zobrazí terminálové okno systému OSMRR, kde je uživatel vyzván k zadání uživatelského jména (heslo není požadováno), viz *Obrázek 1*.

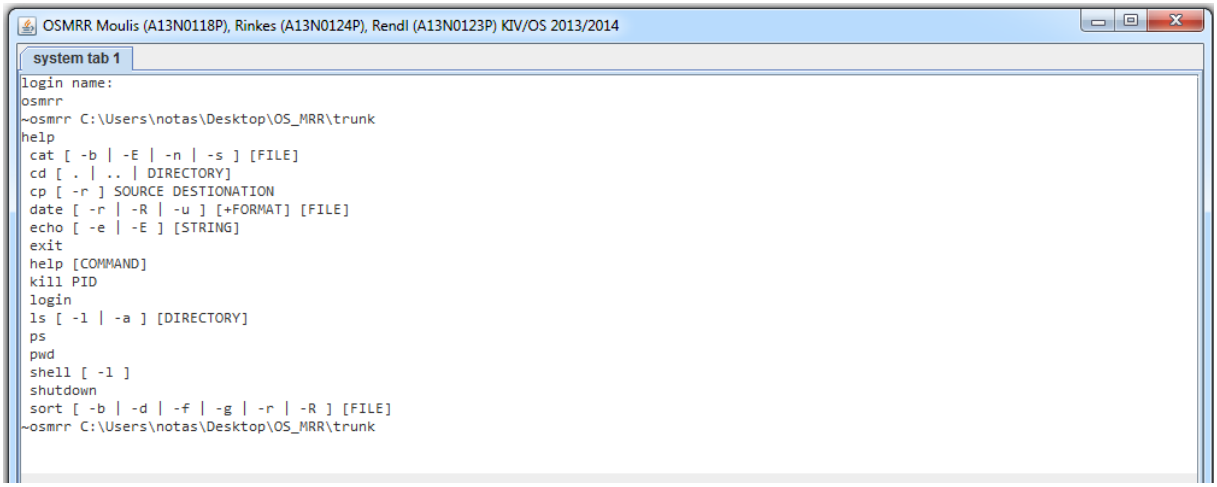


Obrázek 1: Přihlášení uživatele

2.4 Ukázka obsluhy shellu

2.4.1 Příkaz HELP

Samotný příkaz *HELP* vypíše stručné informace o všech dostupných programech (název programu, parametry a argumenty), viz *Obrázek 2*.



Obrázek 2: Ukázka příkazu HELP

Poznámka:

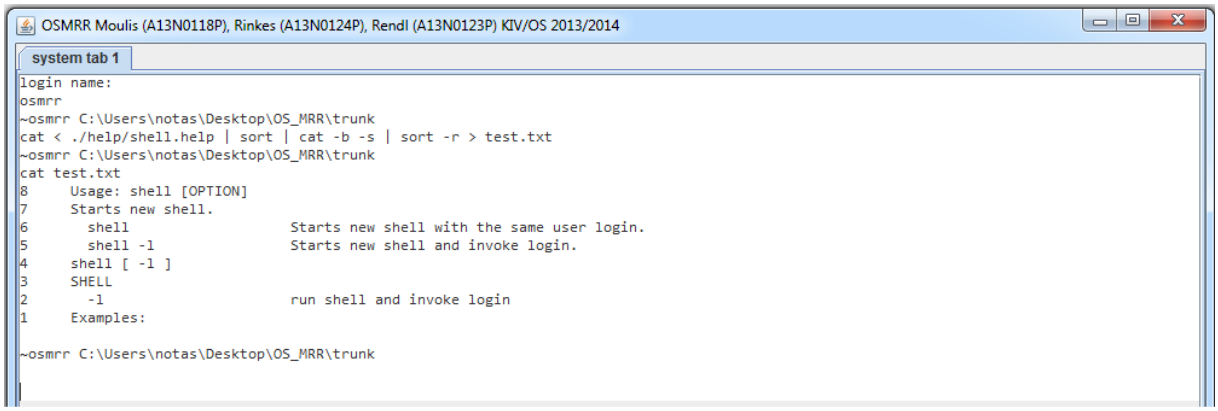
Příkaz 'HELP název_programu' vypíše detailní informace o programu.

2.4.2 Přesměrování I/O, řetězení programů, použití parametrů a argumentů

Na následujícím příkladu je ukázáno, jak zapsat složitější příkaz, který obsahuje přesměrování souboru na vstup programu '<', přesměrování výstupu z programu do souboru '>', zřetězení programů '|'. Zároveň je zde ukázáno jak pracovat s parametry '-...' a argumenty programů (informace o parametrech a argumentech programů viz *2.4.1*).

Příkazy:

```
cat < ./help/shell.help | sort | cat -b -s | sort -r > test.txt
cat test.txt
```



Obrázek 3: Výsledek ukázky příkazů v shellu

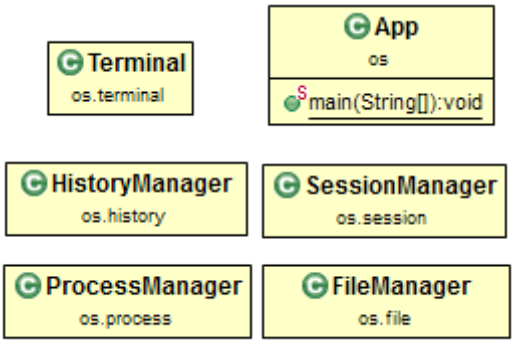
Poznámka:

Parametry programu musí následovat bezprostředně za příkazem a argumenty je možno zapsat až za parametry programu, např. 'CAT -b -s soubor.txt', ale nikoli ~~'CAT soubor.txt -b -s'~~.

3 Programátorský manuál

3.1 Jádro

Jádro aplikace se skládá z manažerů. Manažeři jsou navrženi podle návrhového vzoru singleton, aby v jádře mohl existovat opravdu jen jeden od každého. *FileManager* je manažer pro práci s soubory a souborovým systémem. *SessionManager* je manažer pro práci s relacemi přihlášených uživatelů. *HistoryManager* je manažer pro načítaní, ukládání a práci s historií příkazové řádky pro jednotlivé relace. *ProcessManager* je manažer pro plánovaní, údržbu, výpis a ukončovaní procesů, které naplánoval a ještě nebyly ukončeny. Třída *App* je main třída pro spuštění.



Obrázek 4: Třídy jádra

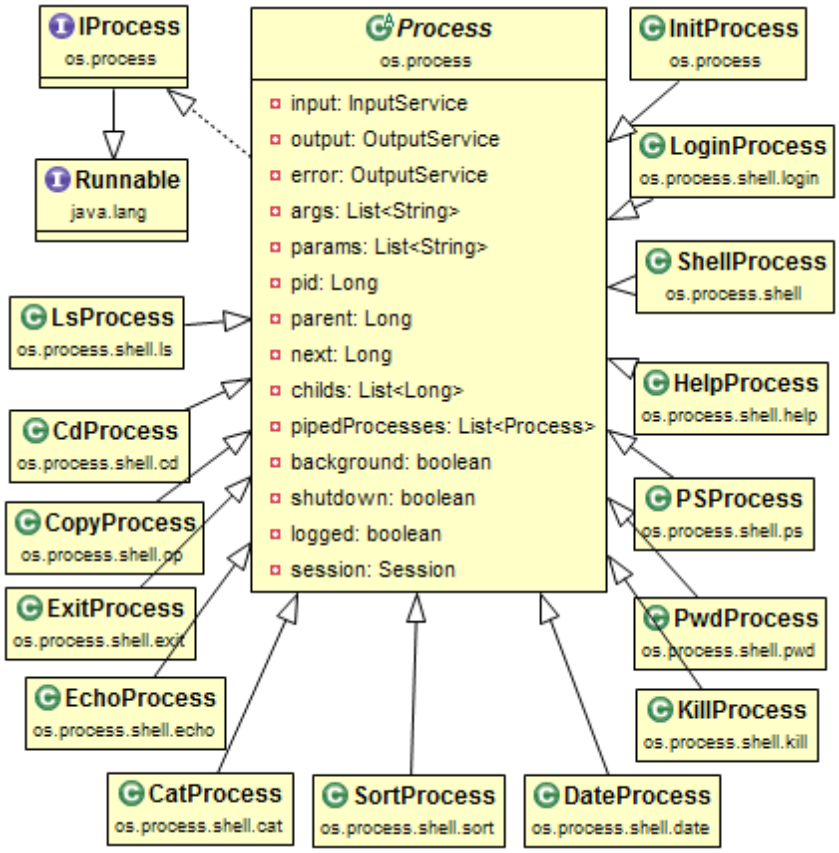
3.2 Spuštění systému

Aplikace po spuštění main třídy vytvoří init proces a terminálové okno a inicializuje logování pomocí Java Logging API. Init proces inicializuje všechny manažery, naplánuje si první shell proces, který spustí a init proces se uspí, dokud proces shell žije. Shell proces si vynutí login pokud není již přihlášen.

3.3 Procesy

Všechny procesy jsou potomci abstraktní třídy *Process* a jsou spouštěny jako samostatná vlákna. Tato třída poskytuje základní strukturu pro jednotnou manipulaci s procesy (pid, rodič, potomci, vstup, výstup, chybový výstup, argumenty, parametry). Zároveň *Process* implementuje rozhraní *Runnable*, aby bylo možné ho použít ve vláknech. O plánování procesů se stará modul jádra *ProcessManager*.

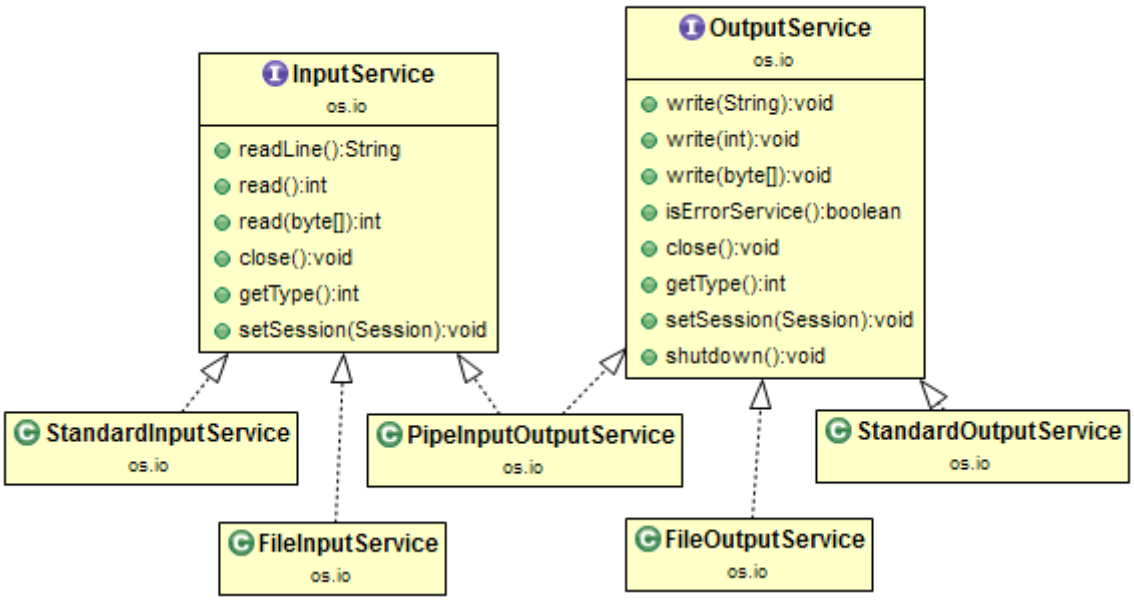
Hierarchie procesů a práce s ní je podobná jako v unix systémech. Proces má svého rodiče a potomka. Pokud je proces ukončen, jeho rodič přebere mezi své potomky potomky ukončeného procesu.



Obrázek 5: UML diagram procesů

3.3.1 Vstup a výstup procesů

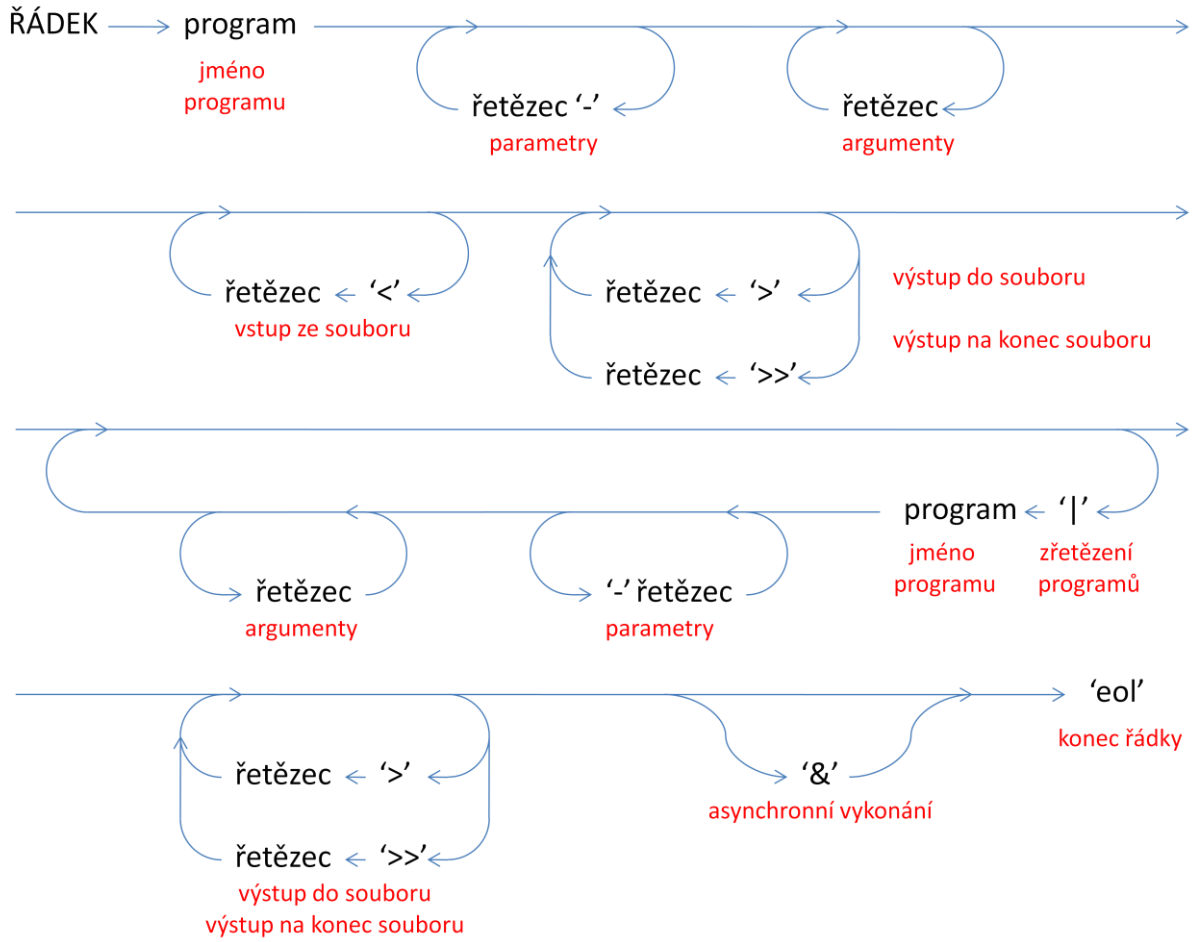
Pro implementaci vstupu a výstupu jsme použili jednotné rozhraní *InputService* a *OutputService*. Z nich byly vytvořeny implementace pro standardní, souborový vstup/výstup a objekt pro propojování procesů pomocí trubky (pipe).



Obrázek 6: UML diagram I/O

3.4 Gramatika

Příkazy zadané do terminálového okna jsou předány syntaktickému analyzátoru. Ten nejprve provede lexikální analýzu vstupu, tj. rozdělí vstup podle mezer na lexémy. Ještě před samotným rozdělením, se provede kontrola, zda vstup obsahuje mezery na správných místech. Pokud ne, mezery se doplní před každý operátor (<, >, |, &) stejně, jako tomu je například v Linuxu v programu bash.



Obrázek 7: Schéma použité gramatiky (řádky na sebe navazují od shora dolů)

Následně sémantický analyzátor zvaliduje, zda vstup odpovídá námi navržené gramatice, viz *Obrázek 7*. Jedná se konečný automat, sestrojený podle gramatiky, který rozpoznává lexémy na základě porovnávání, zda se může daný lexém nacházet na určité pozici ve vstupu.

Při procházení vstupního řetězce analyzátor vytváří seznam programů *List<Process> listOfProcesses*, nastavuje parametry/argumenty programům, přesměruje vstupy/výstupy do/ze souborů, zřetězí programy a nastaví příznak, zda programy poběží na pozadí. Pokud jsou všechny lexémy validní a není chyba v syntaxi, analyzátor vrátí seznam programů shellu, který následně programy spustí. V opačném případě je uživatel informován o tom, že je ve vstupním řetězci chyba a kde se tato chyba nachází.

4 Závěr

Semestrální práce splňuje standardní zadání. Program simuluje chování operačního systému, tj.: poskytuje služby programům, plánuje/spouští/ukončuje programy, zajišťuje prostředky pro komunikaci mezi programy a prostředky pro práci se soubory.

Shell obsahuje všechny povinné příkazy (příkaz *MAN* je implementován jako příkaz *HELP*). Navíc jsme implementovali příkazy COPY a DATE. Dále prompt shellu vždy obsahuje celou cestu k aktuálnímu adresáři, ve kterém se uživatel nachází. Historie příkazů společně s adresářem, kde byl příkaz spuštěn, je ukládána do textového souboru pro každého uživatele zvlášť do složky *history*. Zde je pro každý login textový soubor '*history_login.hist*'. A programy je možné spustit na pozadí pomocí přepínače &.

Podmínkou pro úspěšnou validaci zadávaných příkazů je správné seřazení parametrů a argumentů. Parametry programu musí vždy následovat bezprostředně za jménem programu a argumenty až za parametry. Všechny programy, které pracují se soubory (přesměrování vstupu/výstupu, soubory zadané jako argumenty), dokáží pracovat s absolutními i relativními cestami.

Soubory s nápovědou pro jednotlivé programy jsou umístěny ve složce help kvůli jednoduché editaci. Zde je pro každý program textový soubor '*program.help*', který na první řádce obsahuje stručný popis použití programu (parametry, argumenty) a dále obsahuje detailní popis funkcionality, význam jednotlivých parametrů nebo argumentů a ukázku použití na jednoduchých příkladech.

Funkcionalitu programu jsme testovali především na operačních systémech Windows 7 a 8, ale otestovali jsme i běh programu na Linuxových distribucích Ubuntu, Debian a OpenSuse. Některé příkazy (CD, LS) zde ale nemusí být plně podporovány, např. pokud není program spuštěn s právy super uživatele, ne všechny složky budou přístupné.

Rozdělení práce mezi členy týmu:

Jan Moulis - implementace syntaktického analyzátoru, procesů

Kamil Rendl - implementace procesů, testování

Jakub Rinkes - implementace jádra, kostry procesů, GUI, historie

V průběhu tvorby semestrální práce jsme přemýšleli, jak by šlo toto zadání "Simulace operačního systému v Javě" do budoucna vylepšit a došli jsme k závěru, že by nebylo špatné mít možnost použít pro sestavování programu Ant scripty nebo Maven. V tomto případě se sice nejedná o nijak velký projekt, ale Ant i Maven nabízí určité zjednodušení práce oproti použití dávkových souborů a navíc jsou multiplatformní. Dalším výhodou by bylo povolit použití vyšší verze Javy, ale chápeme proč je zde podmínkou pouze Java ve verzi 1.5.0.