Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra informatiky a výpočetní techniky

Správce virtuálních strojů

Týmová semestrální práce z předmětu

Operační systémy

Kontakt na tým:

kalwi@students.zcu.cz

Členové týmu:

Natalia Rubinova, A080082P

Martin Sloup, A08N0111P

Jiří Kučera, A08N0092P

Zadání

Cílem semestrální práce je vytvořit model správce virtuálních strojů, který bude simulovat prostředí pro běh procesů, umožní jejich spuštění a ukončení, poskytne vstupy a výstupy, podpůrná API a z uživatelského hlediska použitelné rozhraní založené na textovém shellu.

Uživatelský manuál

Pro spuštění aplikace je vyžadováno JDK 5.0 nebo vyšší.

Aplikaci je před spuštěním nutno zkompilovat, to se provede spuštěním scriptu:

```
compile.bat
```

Script vygeneruje soubor os.jar, který je možno spustit souborem:

```
run.bat
```

Následně se otevře konzolové okno a uživatel je požádán o zadání uživatelského jména. Po zadání se uživatel ocitne v shellu, kde může zadávat následující příkazy: cat, clear, echo, eightball, kill, ls, man ps, pstree, pwd, receive, send, shell, shutdown, sort. Podrobná nápověda k příkazům se vypíše příkazem man.

V rámci konzole lze otevřít nové konzolové okno zkratkou CTRL+N. Konzoli je kromě standardních ovládacích prvků okna možné ukončit i klávesovou zkratkou CTRL+Q. V rámci konzolového okna je možné použít historii zadaných dat (řádků). Klávesami šipka nahoru a šipka dolu se prochází položkami historie. V aplikacích, které čtou ze standardního vstupu, lze tento vstup uzavřít klávesovou zkratkou CTRL+D.

Většinu obsluhy operačního systému uživatel stráví v aplikaci shell. Kromě postupného spouštění jednotlivých příkazů je možné příkazy řetězit do kolony znakem roura (znak „|“) a přesměrovávat vstup a výstup této kolony znaky menší a větší (znaky „<“ a „>“).

Pár příkladů:

```
C:\>cat soubor.txt | sort
C:\>sort <netridene.txt >tridene.txt
```

Samozřejmě je možné používat přesměrování vstupů / výstupů společně s řetězením příkazů:

```
C:\>echo <netridene.txt | sort >tridene.txt
```

Programátorský manuál

Aplikace je členěna na čtyři logické celky: manažer virtuálních strojů, terminálové okno, virtuální uživatelské aplikace a textový shell (přestože se jedná též o uživatelskou aplikaci, kvůli rozsahu je pro něj vyčleněna samostatná kapitola). Tyto celky jsou vhodně uspořádány do balíků.

Manažer virtuálních strojů

- balík cz.zcu.kiv.os.manager

Manažer (třída VMManager) tvoří „jádro“ celého virtuálního operačního systému. Hlavní myšlenkou návrhu manažeru je to, že neběží ve vlastním samostatném vlákně, ale poskytuje služby virtuálním procesům, v jejichž vlákně je pak vykonává:

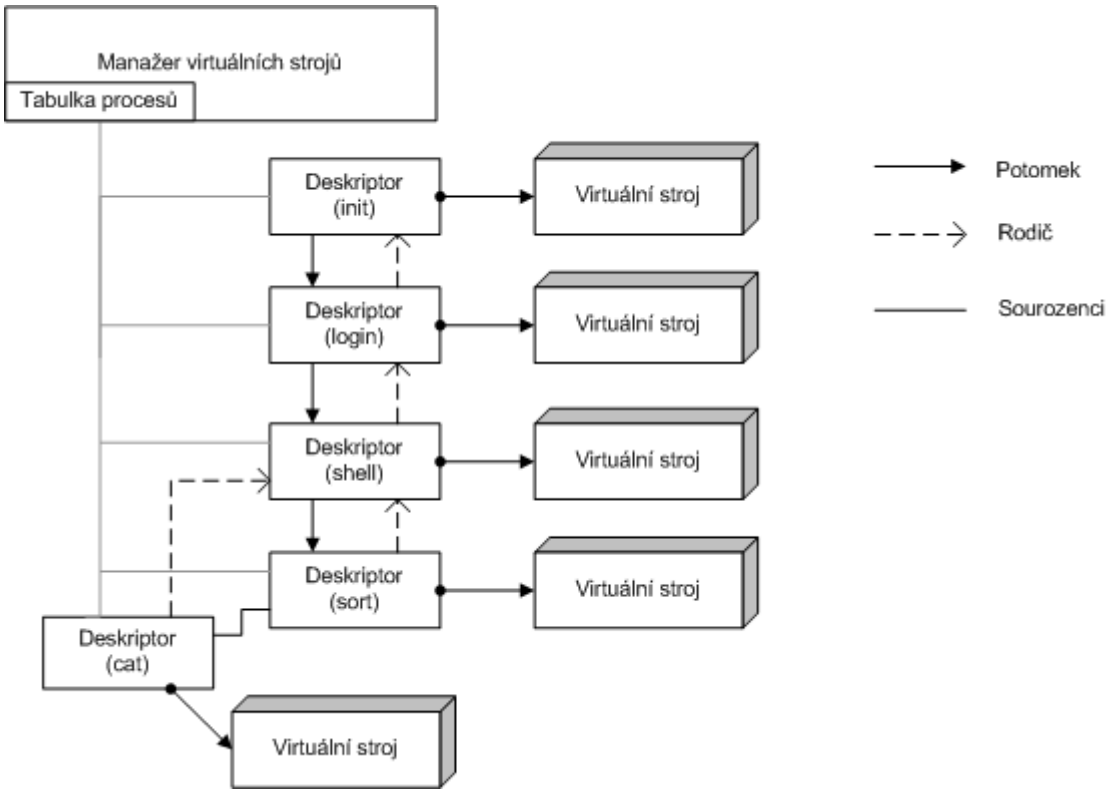
- vytvoření virtuálního stroje (resp. procesu) – vytvoří deskriptor procesu, který zařadí do datových struktur (tabulka a hierarchický strom procesů), vytvoří nový virtuální stroj, do kterého class loaderem načte uživatelskou aplikaci a spustí ji jako virtuální proces.
- odstranění deskriptoru – odstraní deskriptor skončeného procesu z datových struktur, uzavře jeho vstupy a výstupy a informuje o tom rodiče a sourozence.
- spuštění procesu – spustí všechny potomky zadaného procesu.
- zabití procesu – ukončí proces a všechny jeho potomky.
- vypnutí virtuálního operačního systému – ukončí všechny běžící procesy, čímž se zároveň ukončí virtuální operační systém.

Další komponenty, které jsou součástí manažeru:

Deskriptor procesu (tř. VMDescriptor) zastupuje proces v datových strukturách a obsahuje důležité informace o procesu: název procesu, argumenty procesu, PID, uživatelské jméno vlastníka procesu, odkaz na rodiče, potomka a následujícího sourozence, vstup, výstup a odkaz na samotný proces (resp. virtuální stroj).

Virtuální stroj (tř. VirtualMachine) je běhové prostředí virtuálního procesu. Java bohužel neumožňuje zasahovat do běhu vlákna podobně jako skutečný operační systém do běhu procesů, proto je třeba alespoň základní funkcionalitu napodobit. Ve virtuálním stroji je kromě vlákna samotného procesu spuštěno druhé, servisní vlákno, které čeká na dokončení procesu, potomků a sourozenců a poté se postará o uzavření vstupů a výstupů a zrušení deskriptoru procesu. Třída také poskytuje API jak pro běžící proces, tak pro ovlivňování běhu procesu zvenčí, chová se tak jako aplikační rozhraní mezi jádrem a uživatelskými aplikacemi. Nejdůležitější metody:

- spuštění procesu nebo kolony procesů – důležité argumenty jsou pole procesů a jejich argumentů a vstup a výstup do/z kolony (více viz popis rour).
- tisk na standardní výstup
- čtení ze standardního vstupu
- zjištění a nastavení aktuálního adresáře
- ukončení procesu
- volání služeb manažeru (zabití procesu, vypnutí stroje)



Obrázek 1: Schéma organizace manažeru virtuálních strojů

Jsou-li uživatelské aplikace spuštěny v koloně, virtuální stroj, který kolonu spouští, jejich vstupy propojí rourami (tř. Pipe). Tato třída má pouze vstup a na něj napojený výstup – oba tyto je pak možno předat procesu jako standardní vstup a výstup. Kromě toho ale byla zobecněna tak, že lze její vstup či výstup napojit na soubor a použít ji tak i v případě přesměrování vstupu/výstupu do souboru.

- balík cz.zcu.kiv.os.manager.mailbox

Tento balík obsahuje třídy související s možností posílání zpráv mezi uživateli. Především se jedná o třídu MailBox, která obsahuje metodu k přístupu do uživatelovy poštovní schránky (tř. UserMailBox). Poštovní schránka uživatele pak obsahuje metody pro přidání nové zprávy (reprezentovaná třídou MailBoxItem) do uživatelovy schránky a načtení zprávy ze schránky.

Uživatelské aplikace

- balík cz.zcu.kiv.os.bin

Uživatelské aplikace jsou koncipovány jako samostatné nezávislé třídy – jádro neví, jaké aplikace jsou přítomny, a snaží se je podle jejich názvu dynamicky načíst class loaderem. To umožňuje libovolně přidávat další aplikace, aniž by bylo třeba jakkoliv zasahovat do zbytku kódu. Jedinou výjimkou jsou hierarchické závislosti kořenových procesů init, login a shell, jejichž názvy jsou uloženy v konfigurační třídě cz.zcu.kiv.os.Configuration.

Každá aplikace dědí třídu virtuálního stroje, a musí implementovat jedinou abstraktní metodu main(), která běží ve vlákně procesu po jeho spuštění.

Je-li některá aplikace implementována více třídami, umístí se tyto třídy do vlastního podbalíku umístěného v balíku s aplikacemi, a to z toho důvodu, aby se class loader nepokoušel tyto třídy načíst jako uživatelské aplikace. Spouštěcí třída je samozřejmě umístěna ve výše uvedeném balíku.

Shell

- balík cz.zcu.kiv.os.bin.shell

Aplikace shell (tř. cz.zcu.kiv.os.bin.Shell) tvoří pro operační systém příkazovou řádku. Načítá uživatelem zadaný vstup (řetězec s příkazy). Tento vstup následně předá parseru, který vrátí seznam (kolonu) procesů, jejich argumenty a vstup a výstup kolony. Voláním API virtuálního stroje se vytvoří a spustí příslušné procesy. O propojení vstupů a výstupů procesů pomocí rour a přesměrování vstupu a výstupu kolony na patřičná zařízení (disk, klávesnice, obrazovka) se pak postará virtuální stroj. Po spuštění procesů shell čeká na dokončení běhu těchto procesů. Kromě spouštění dalších aplikací umožňuje také spouštět vlastní vestavěné příkazy. Mezi ně například patří příkaz cd, který voláním metody virtuálního stroje mění aktuální pracovní adresář procesu, nebo příkaz exit sloužící k ukončení shellu. Další funkcionalitou shellu je možnost automatického přijímání zpráv, které byly uživateli zaslány příkazem send. Přijaté zprávy jsou vypsány vždy před vstupem uživatele.

Shell je rozsáhlá aplikace, proto je pro něj vytvořen další podbalík, ve kterém se nachází parser příkazů zadávaných v shellu. Samotný shell je umístěn v balíku společně s ostatními uživatelskými aplikacemi operačního systému.

Parser vstupu z příkazové řádky (tř. Parser) se snaží rozpoznat bloky podle gramatiky, jež je popsaná níže. Zároveň se provádí syntaktická analýza řádku. Pokud se narazí na problém, dojde k vyhození výjimky a ukončení procesu parsování. Ve výjimce je obsažena informace o pozici, kde nastala chyba. Speciální vlastností parseru je podpora uvozovek a escape sekvencí (zadávaných zpětným lomítkem). Výstupem parseru je kolona procesů, jejich vstupní parametry a případné přesměrování vstupu a výstupu.

Gramatika parseru

Gramatika, podle které parser rozpoznává bloky příkazu:

ŘÁDEK → PŘÍKAZ VSTUP {‘|’ PŘÍKAZ} VÝSTUP
PŘÍKAZ → JMÉNO { ARGUMENT }
JMÉNO → *řetězec*
ARGUMENT → *řetězec*
VSTUP → e | ‘<’JMÉNO
VÝSTUP → e | ‘>’JMÉNO

Terminálové okno

- balík cz.zcu.kiv.os.console

Okno terminálové konzole (tř. ConsoleWindow) představuje jednoduchou implementaci vstupně-výstupního uživatelského rozhraní operačního systému. Kromě zadávání vstupu z klávesnice a vypisování výstupu na obrazovku umožňuje použití zkratek pro vytvoření nového okna konzole a uzavření standardního vstupu (vstupu z klávesnice). Také obsahuje podporu historie zadaných řádků z klávesnice (tř. ConsoleHistory).

InputStream konzole (tř. ConsoleInputStream) implementuje abstraktní metody třídy InputStream, čímž do třídy ConsoleWindow přináší podporu výstupu konzole (tedy vstupu programu, též vstupu z klávesnice) a následné zapouzdření pomocí již existujícího API Javy dále v programu.

OutputStream konzole (tř. ConsoleOutputStream) kromě implementace abstraktních metod třídy OutputStream (výstupu na konzolové okno) obsahuje metody pro přístup ke specifickým vlastnostem třídy ConsoleWindow, jako jsou například metody vracející velikost okna ve znacích. Ty jsou využity při formátování výstupu z určitých příkazů operačního systému.

Historie příkazů konzolového okna (tř. ConsoleHistory) uchovává seznam již zadanýchřádků do konzolového okna a pomocí jednoduchého rozhraní umožňuje jednoduchou manipulaci s tímto seznamem.

Závěr

Realizovali jsme všechny povinné body zadání. Byť se jedná jen o model, snažili jsme se co nejvíce se přiblížit skutečnému systému. Procesy jsou kromě tabulky procesů organizovány v hierarchickém stromu sourozenců v koloně, rodičů a potomků. Kladen byl důraz na bezpečnost, např. API virtuálního stroje je napsáno tak, aby si procesy mezi sebou nemohly navzájem měnit své parametry.

Mezi rozšíření oproti zadání patří:

- víceuživatelský systém (pro jednoduchost a snadnou demonstraci funkčnosti aplikace ale nejsou ošetřena uživatelská práva např. při zabíjení procesů jiných uživatelů)
- prompt ukazuje celou cestu k aktuálnímu adresáři
- podporován je adresář ~ jako domácí adresář
- podpora zasílání zpráv mezi uživateli
- vlastní terminálová konzole
- další příkazy shellu / uživatelské aplikace:
    - clear – smazání textu v konzoli
    - debuginfo – zobrazí některé ladicí informace
    - eightball – jednoduchá textová hra pro účely ladění
    - pstree – zobrazí běžící procesy v hierarchickém stromu
    - receive – čeká na přijetí zprávy od jiného uživatele
    - send – odešle zprávu jinému uživateli
- dokumentace v angličtině

Nedostatky a možná vylepšení:

Implementace skupin úloh (jobs) by kromě možnosti spouštění procesu na pozadí umožnila i paralelní spouštění více potomků tak, aby nemusely čekat na dokončení svých sourozenců.

Zároveň s implementací jobs by po zabití procesu všechny jeho potomci mohli pokračovat ve své činnosti. Bez této funkcionality je běh potomků zabitého procesu, resp. restrukturalizace stromu procesů příliš složitá, takže se ukončuje celý podstrom potomků.

Vstup a výstup z konzole se pro procesy, které mají potomky, neuzamyká, takže rodiče musejí čekat na dokončení potomků místo toho, aby případně mohly pokračovat v práci, která vstup a výstup

nevyžaduje. Taktéž by bylo vhodnější, kdyby vstupy a výstupy měly počítadlo procesů, které je využívají – při dosažení nulového počtu „klientů“ by se vstup/výstup uzavřel. Současná implementace funguje tak, že vstup/výstup je uzavřen procesem, který jej použil jako první a od nějž všichni potomci tento vstup/výstup dědí (pro ně se uzavření I/O jen emuluje).

Rozdělení práce mezi členy týmu:

Jiří Kučera – správce virtuálních strojů a související komponenty, administrativní záležitosti

Martin Sloup – shell, terminálová konzole, podpora zasílání zpráv mezi uživateli, správa SVN serveru

Natalia Rubinova – uživatelské aplikace, ladění, dokumentace v angličtině

Přestože se jedná jen o demonstrační „šuplíkovou“ aplikaci, její přínos pro členy týmu určitě není zanedbatelný. Nejvýznamnějším přínosem je procvičení práce s vlákny, zjištění možností a nedostatků práce s vlákny v jazyce Java, dále pak prohloubení znalostí principů fungování operačního systému (především správy procesů). A nakonec každá týmová semestrální práce přispívá k rozvoji schopností práce v týmu.