Simulátor OS

2014

Západočeská univerzita v Plzni   
Fakulta aplikovaných věd  
Katedra informatiky a výpočetní techniky  
Předmět: Operační systémy (KIV/OS)

Jan Bláha, A14N0118P

Radek Bouda, A14N0120P

David Steinberger, A13N0095P

Kontakt: rbouda@students.zcu.cz

Zadání

Simulátor operačního systému

Cílem práce je navrhnout a implementovat model operačního systému. Základem modelu operačního systému bude shell, který bude mít implementovány základní příkazy jako cat, cd, echo, exit, kill, ls, man, ps, pwd, shell (rekurzivní spouštění shellu), shutdown a sort, s podporou ukládání příkazů.

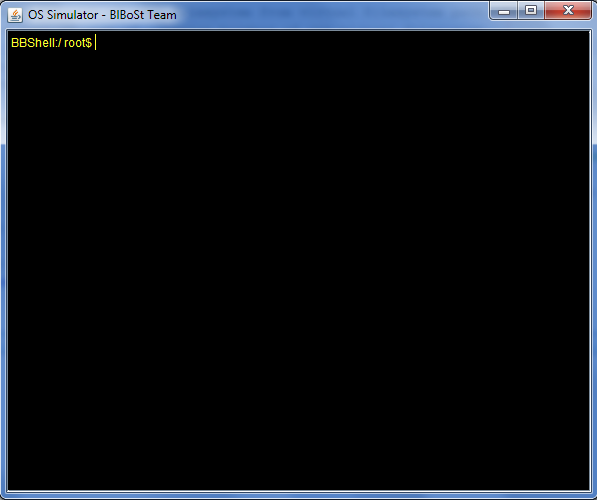
Musí být možné jednotným způsobem získat nápovědu ke každému příkazu.

Vzorem pro shell a příkazy bude linuxový bash.

Uživatelská dokumentace

Námi navržený operační systém je implementován v jazyce Java, proto je nutné mít na počítači, kde je program spouštěn, nainstalovanou Javu verze 7 a vyšší.

Operačním systém lze zkompilovat pomocí souboru Compile.bat a spustit dvojím kliknutím levého tlačítka myši na soubor Run.bat. Okamžitě se zobrazí terminálové okno (viz Obrázek 1), které slouží jako standartní vstupní i výstupní grafické rozhraní operačního systému.



Obrázek 1

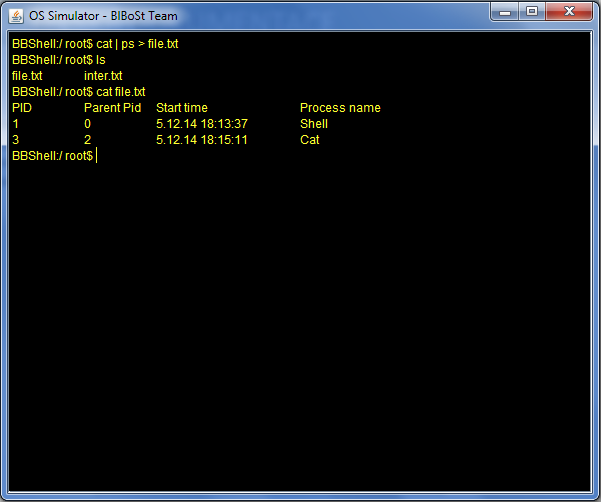
Při prvním spuštění operačního systému se Vám vytvoří vlastní kořenový adresář *filesystem* pro náš operační systém. Tento adresář se nachází ve stejném adresáři, jako se nachází soubor Run.bat, kterým jste operační systém spustili. Operační systém má přístup pouze k adresářům a souborům umístěným   
v tomto adresáři. Při spuštění Operačního systému se nacházíme přímo v adresáři *filesystem*, tudíž   
v kořenovém adresáři.

Pokud máme aktivní okno operačního systému, můžeme pomocí klávesnice zadávat příkazy shellu. Příkazy můžeme také vkládat pomocí klávesové zkratky Ctrl+V.

Příkazy zadáváme ve formátu: Příkaz { " | " Příkaz} (" < " Vstup) (" > " Výstup)

Slovem Příkaz je myšleno název procesu, který chceme spustit a všechny jeho parametry. Závorky { a } nám dávají najevo, že hodnota v nich uvedená může být použita 0x až nekonečně mnohokrát. Nový příkaz následující znak " | " je tedy možný, zároveň není povinný, pokud chceme spustit pouze jeden proces. Závorky ( a ) nám dávají najevo, že hodnoty v nich uvedené můžeme použít, či nemusíme. Avšak v případě použití je můžeme použít pouze jednou. Přesměrování vstupu a výstupu musí být na konci příkazu.

Hodnoty Vstup a Výstup nejsou povinné, a pokud nejsou uvedeny, bude zadanému procesu přiřazen standartní Vstup respektive Výstup. Pokud bychom chtěli specifikovat zvláštní vstup (například vstupní soubor), musíme použít znak " < " před vstupním souborem. Pro výstup pak používáme znak " > ". Příklad syntaxe při použití dvou procesů v pipe a výstupního souboru můžeme vidět ne Obrázku 2.



Obrázek 2

Prvním příkazem vytvoříme výstupní soubor *file.txt* do kterého pomocí procesu *Cat* zapíšeme výstup z procesu *Ps*. Druhým příkazem vypíšeme všechny soubory v adresáři. Zde vidíme nový soubor *file.txt* a již existující soubor *inter.txt*. Poté opět pomocí procesu *Cat* vypíšeme obsah souboru *file.txt*. Soubor *file.txt* zde slouží jako parametr procesu *Cat*, tudíž není potřeba použít znak " < ".

Při použití procesu, který neexistuje, nebo není operačním systémem podporován, vypíše nám terminál hlášku: "-BBShell: SpatnyNazev is not a valid process!".

Při použití existujícího procesu avšak se špatným formátem parametrů tohoto procesu, se nám zobrazí na Terminálu manuál k používání zadaného procesu.

Pro zobrazení stránek manuálu k jednotlivým procesům, použijeme proces *Man*, kde jako první parametr uvedeme název procesu, ke kterému chceme zobrazit manuálové stránky. Manuálové stránky také můžeme také zobrazit použitím jména procesu s parametrem --help (např. Ls --help).

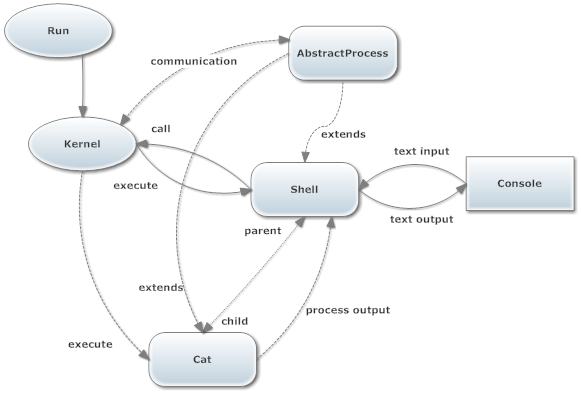
Terminál nám také dovoluje používat hledání mezi příkazy, které jsme již zadali. Pokud v terminálovém okně stiskneme tlačítko šipky nahoru, nebo dolů, můžeme procházet historii použitých příkazů. Šipkou nahoru zobrazujeme starší příkazy. Šipkou dolů poté příkazy novější.

Operační systém je schopný také zaznamenávat několik klávesových zkratek. Již byla zmíněna zkratka *Ctrl+V* pro vkládání textu. Dále můžeme použít zkratku *Ctrl+C* pro okamžité ukončení právě běžícího procesu a *Ctrl+D* pro ukončení standartního vstupu.

Pro ukončení operačního systému použijeme příkaz *shutdown,* pro ukončení právě běžící konzole příkaz *exit*.

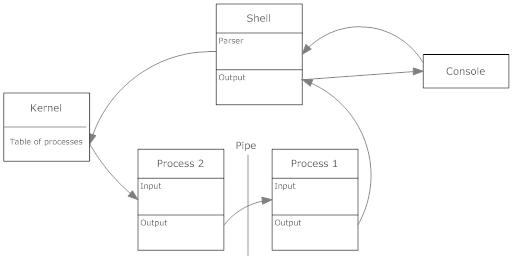
Programátorská dokumentace

Námi navrženou strukturu operačního systému je možné vidět na Obrázek 3. Takto vypadá struktura pro spuštěný Shell a jeden proces.



Obrázek 3

Po spuštění jádra spustíme proces Shell, který vyvolá terminálové okno. S tímto oknem poté komunikuje pouze proces Shell, který tento terminál spustil. Předávání informací mezi procesy a užívání pipe je znázorněno na Obrázek 4. Každý proces může zavolat jádro při požadavku na tabulku procesů. Aktuální umístění ve filesystemu bychom poté hledali v procesu Shell. Složka filesystem je vytvořena v místě uložení spouštěcího souboru, pokud neexistuje, je automaticky vytvořena. Operační systém může pracovat se soubory složkami pouze v této složce.



Obrázek 4

Komunikace v celém systému je následující: Terminál plní uživatelským vstupem výstupní pipe, která je napojená na vstupní pipe Shellu. Shell tento vstup přečte a zanalyzuje. Poté požádá Jádro o vytvoření a spuštění prvního (poslední v sekvenci příkazů) procesu. V případě, že se jedná o vestavěný proces Shellu, vykoná tento proces sám Shell. Tento proces naváže svojí výstupní pipe na vstupní pipe Shellu a rekurzivně volá další procesy (předchozí v sekvenci příkazů). Tyto procesy jsou mezi sebou také navázány pomocí pipes. Poslední vytvořený proces začne vykonávat svojí činnost a plnit výstupní pipe. V případě, že pro svůj běh potřebuje vstup, může využít přesměrování ze souboru nebo číst ze standardního vstupu[[1]](#footnote-1) (terminál). Jakmile proces začne generovat výstup, následující proces může začít číst a pracovat (některé procesy čekají na kompletní výstup). Procesy jsou blokující pomocí IO operací. Tímto způsobem se dostane výstup až do Shellu, který na základě uživatelského vstupu rozhodne, kam se má výstup zapsat.

Pokud je jako parametr použito slovní spojení v uvozovkách, bere systém celé toho spojení jako jeden parametr.

Proces Shell může být také spuštěn pouze s vstupem. V tomto případě vykoná všechny požadované úkoly a vrátí výstup na terminálové okno, ze kterého byl spuštěn.

Procesy Echo, Cd, Exit a Pwd jsou vytvořené jako vestavěné procesy Shellu. Z programátorského hlediska to znamená, že tyto procesy nejsou samostatné třídy v balíku *process*, ale pouze metody ve třídě *Shell*. Z uživatelského hlediska zde není žádný rozdíl.

Gramatika

Řádek = Příkaz { '|' Příkaz } Vstup Výstup  
Příkaz = Jméno {Parametr}  
Jméno = řetězec  
Parametr = řetězec  
Vstup = null | '< ' Jméno  
Výstup = null | ' > ' Jméno

Závěr

Při tvorbě naší práce jsme postupovali dle předem navržených kroků v zadání na stránkách Courseware ZČU. Zároveň jsme si prošli i Nejčastější chyby a tak věříme, že naše práce by měla splňovat všechny požadavky na tuto práci kladenou.

Kromě zadaných procesů jsme implementovali i proces Grep. Všechny procesy také obsahují manuálové stránky, které lze vyvolat jak příkazem Man, tak i parametrem --help. Věříme, že jsme ošetřili všechny chybové hlášky a vytvořili stabilní systém.

Mezi slabé stránky naší realizace bychom zařadili především to, že nemáme implementovaný proces login. S tím je spojené i to, že náš systém není multi-uživatelský. Při spuštění několika procesů Shell se tyto procesy chovají jako potomci Shellu, ze kterého byli spuštěny. Tudíž při ukončení prvního Shellu ukončíme zároveň všechny běžící procesy.

Jako silnou stránku naší implementace vidíme jednoduchost a uživatelskou použitelnost našeho systému. Uživateli je poskytnuta srozumitelná nápověda i s příklady použití jednotlivých procesů.

Možnosti vylepšení systému vidíme především v multi-uživatelském prostředí a přidání dalších parametrů k již existujícím procesům. Samozřejmě lze práci vylepšit i přidáním dalších procesů a možností usnadnění v používání jako přívětivější grafické rozhraní.

Práce na systému byla rozdělena spíše podle schopností jednotlivých členů týmu, než podle objemu vykonané práce. Radek Bouda, vedoucí týmu, se staral především o programátorskou část a udržování jednotného stylu a srozumitelnost kódu. David Steinberger měl na starosti programátorskou část a působil jako neocenitelný člen týmu jak znalostmi, tak zkušenostmi. Jan Bláha navrhl původní architekturu systému, staral se o syntaktický analyzátor a vytvoření dokumentace a prezentací týmu.

Programování vlastní simulace operačního systému nám přineslo podrobnější znalosti jazyka Java především v práci s Inputy a Outputy, kde jsme byli nuceni upravit knihovnu PipedInputStream a PipedOutputStream. To hlavně z důvodů implementovaného vteřinového Waitu v původní knihovně jazyka Java, který v naší implementaci systému tento Wait způsoboval chybu při zadání více příkazů v rychlém sledu.

Do dalších let bychom byli rádi, kdyby bylo více informací o prezentaci práce v polovině semestru. Například co vše se na prezentaci hodnotí (celková prezentace, množství odvedené práce, vystupování, připravenost na dotazy, vše najednou).

1. Proces vždy požádá o standardní vstup Shell a ten mu přidělí jednu řádku načtenou z terminálu. [↑](#footnote-ref-1)