Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra informatiky a výpočetní techniky

Virtual machines manager

Team semester work in subject

Operating systems

Team contact:

kalwi@students.zcu.cz

Team members:

Natalia Rubinova, A080082P

Martin Sloup, A08N0111P

Jiří Kučera, A08N0092P

Task

The goal of the semester work is to make a model of a virtual machines manager that will simulate environment for processes running, will make possible to start and terminate a process, will provide inputs and outputs, supportive API and useful user interface based on text shell.

User Manual

For launching the application you need JDK 5.0 or higher version.

It is required to compile application before launching it. The compiling is performed by running a script:

```
compile.bat
```

The script will generate os.jar file that could be launched by another file:

```
run.bat
```

The opening of the console window follows, user is requested to enter user name. After entering the name user finds himself inside the shell, where he can enter the following commands: cat, clear, echo, eightball, kill, ls, man ps, pstree, pwd, receive, send, shell, shutdown, sort. Detailed help to the commands is shown by the command man.

Within the console it is possible to open the new console window by using the shortcut CTRL+N. Besides the standard control elements of the window the console can be finished by using the shortcut CTRL+Q. Within the console window the usage of the entered commands history is possible. It is possible to move within the history of items using the arrow-up and arrow-down keys. Closing of standard input can be done by using CTRL+D shortcut in the applications that read from it.

User spends within shell application the most of operating system maintenance. Besides the sequential launching of the single commands it is possible to concatenate the commands to launch them in chain by pipe sign (sign '|') and to redirect input and output of the queue by using signs more and less ('<' and '>')

A couple of examples:

```
C:\>cat file.txt | sort
C:\>sort <unsorted.txt >sorted.txt
```

Of course, the usage of input/output redirecting together with concatenation of commands is possible:

```
C:\>echo <unsorted.txt | sort >sorted.txt
```

Programmers manual

The application is divided into four logical units: virtual machines manager, terminal window, virtual user application and text shell (despite the fact that it is user application only, an independent chapter was dedicated to it because of its size). These units are suitable ordered into packages.

Virtual Machine Manager

- package cz.zcu.kiv.os.manager

The manager (class VMManager) represents a 'core' of the entire virtual operating system. The basic thought of the manager concept is that it does not run within its own thread, but provides services to virtual processes, whose thread they are then executed in:

- creation of virtual machine (or processes) — creates a process descriptor that is put into data structures (process table and hierarchical process tree), it also creates new virtual machine that user application is read by class loader into and starts it as a virtual process
- destroying the descriptor- eliminates descriptor of finished process from data structures, closes its inputs and outputs and notifies a parent and siblings respectively
- launching the processes — launches all child processes of the corresponding process
- killing the process — finishes the process and all his child processes
- shut down the virtual operating system — finishes all running processes, and consequently virtual operational system

Other components that are parts of the manager:

Process descriptor (class VMDescriptor) represents process in data structures and contains important information about process: name, arguments, PID, process owner user name, reference to parent, child and following sibling, input and output and reference to the process itself (Virtual machine).

Virtual machine (class VirtualMachine) represents a running environment of the virtual process. Java unfortunately does not allow affect running thread similarly alike real operating system with process. Therefore it is necessary to simulate at least basic functionality.

Besides the process thread there is the second service thread within the virtual machine is launched. It waits until the process, children and siblings are finished and later takes care of the input and output closing and process descriptor destroying. The class also provides API for running processes as well as for influence of the running process externally, acts as application interface between core and user applications. The most important methods:

- launching the process or processes chain — important arguments are field of processes and their arguments, input and output to/from chain (more in part of pipe description)
- printing to standard output
- reading from standard input
- getting and setting up of the actual directory
- finishing the process
- calling of the manager services (process killing, machine shutdown)

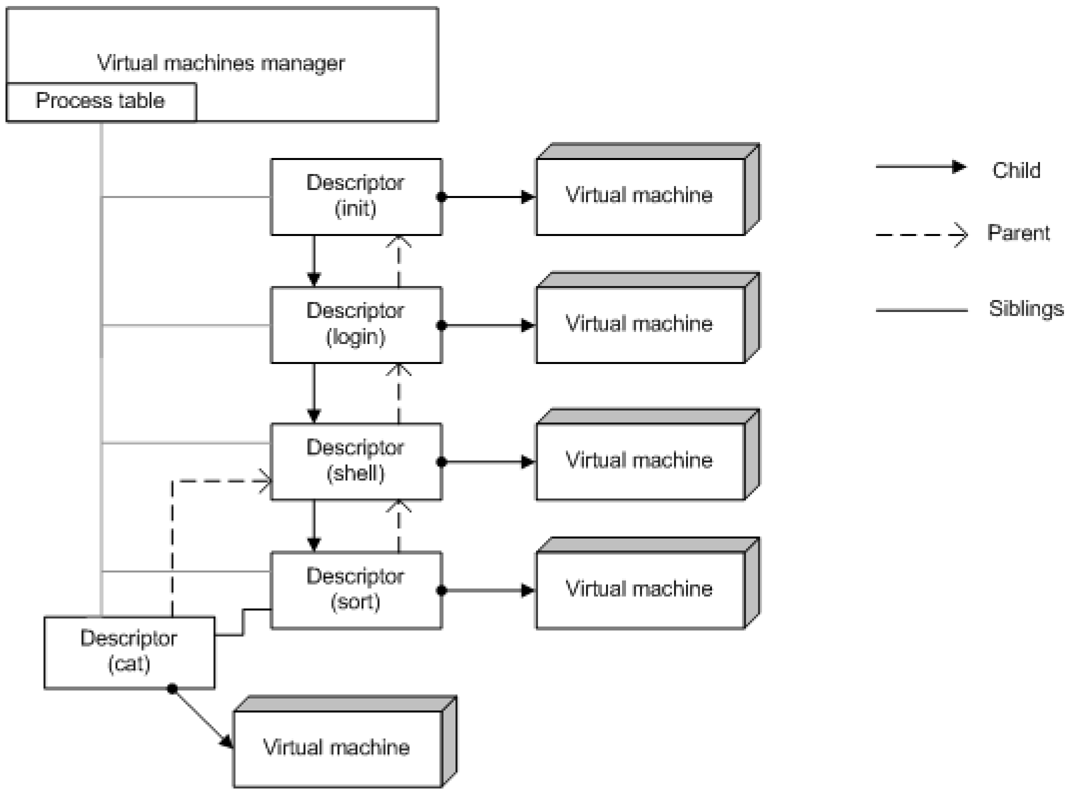

Figure 1: Virtual machines manager organization scheme.

When user applications are launched in a chain, virtual machine launching the chain connects their inputs through pipes (class Pipe). This class has only input and output connected to it — both later could be passed to the process like standard input and output. Besides, it was generalized the way that it is possible to connect to the file either its input or its output and to use it that way in case of redirecting of input/output to file.

- package cz.zcu.kiv.os.manager.mailbox

This package contains classes related to the ability of sending messages between users. Above all it concerns MailBox class that contains the method allowing access to the user mailbox (class UserMailBox). User mailbox in its turn contains methods allowing new message adding to user mailbox (represented by MailBoxItem class) and message reading from the mailbox.

User applications

- Package cz.zcu.kiv.os.bin

User applications are developed as single independent classes: core does not know which applications are present and tries to load them according to their names dynamically by class loader. That allows adding other applications arbitrarily, without any necessity to interfere into the rest of the code. The hierarchical dependencies are only exception of the root processes: init, login a shell, their names are stored in the configuration class cz.zcu.kiv.os.Configuration.

Each application inherits the virtual machine class and must implement only the abstract method main() that runs within the process thread after its launching.

If an application uses more than one class, these classes are placed into its own subpackage located in package with applications. This is in order to class loader does not try to load these classes as user applications. Launching class is obviously placed into above mentioned package with other applications.

Shell

- package cz.zcu.kiv.os.bin.shell

Shell application (class cz.zcu.kiv.os.bin.Shell) creates a command line for operating system. It reads entered user input (command string). This input is passed to the parser that returns the list (chain) of the processes, their arguments and input and output of the chain. Corresponding processes are created and launched by calling the virtual machine API. The connection between inputs and outputs of the processes with pipes as well as chain input and output redirection to the appropriate devices (disk, keyboard and screen) is managed by virtual machine. After launching the processes shell waits until running processes are finished. Besides of the launching other applications shell allows to launch its own built-in commands, for example command cd which changes current process working directory by calling virtual machine method, or command exit that serves for shell finishing. Another shell functionality is represented by option of automatic receiving of message sent to the user by send command. Received messaged are always written before users input.

Shell is the most extensive user application therefore another subpackage is created for it. It contains some parser classes. The shell itself is placed within the package together with the other user application of the operating system.

Parser of command line input (class Parser) tries to identify blocks according to grammar described below. Syntactic analysis of the line is executed simultaneously. In case that a problem occurs, the exception is thrown and the parsing process is finished. The exception contains information regarding the position where the error has occurred. Special parser feature is the quotes and escape sequence (entered by backslash) support. Parser output is chain of the processes, their arguments and eventual input and output redirection.

Parser grammar

Grammar used by parser to identify blocks of commands:

LINE → COMMAND INPUT {'|' COMMAND} OUTPUT
COMMAND → NAME { ARGUMENT }
NAME → *string*
ARGUMENT → *string*
INPUT → e | '<'NAME
OUTPUT → e | '>'NAME

Terminal window

- Package cz.zcu.kiv.os.console

Terminal console window (class ConsoleWindow) represents simple implementation of the operating system input-output interface. Besides input entering from keyboard and writing output to the screen it allows to use keyboard shortcuts for creating new console window and standard input closing (keyboard input). It also supports history of lines entered from keyboard (class ConsoleHistory).

InputStream console (class ConsoleInputStream) implements abstract methods of InputStream class, and thereby it makes possible for ConsoleWindow class to support console output (and thus program and keyboard input) and consequently to encapsulate further in program by existing Java API.

OutputStream console(class ConsoleOutputStream) besides implementation of the abstract methods of OutputStream class (outputs to console window) includes methods for an access to the specific properties of the ConsoleWindow class, like for example methods returning the size of the window in characters. These ones are used for formatting of the output from the specific commands of operating system.

The history of console window commands (class ConsoleHistory) stores list of the lines entered into the console window and simple interface allows simple manipulation with this list.

Conclusion

We have realized all obligatory parts of the task. Though it was supposed to be a model only, we did our best to approach a real system. Processes are organized not only in a table, but as well in hierarchical tree of siblings in chain, parents and children. We put our emphasis to safety and reliability, for example API of virtual machine is developed in such a way that the processes could not change the parameters to another one.

Among the extensions to the main task there belongs the following:

- multiuser system (in order to simplify and demonstrate functionality of the application the user rights, for example while killing the process of other users, are not implemented)
- prompt shows the whole path to the actual directory
- the directory ~ is supported as the home directory
- support of sending message between users
- own terminal console
- other shell / user application commands:
  - clear – clears text in the console
  - debuginfo – displays some debug information
  - eightball – simple text game for debugging
  - pstree – shows running processes as a hierarchical tree
  - receive – waits for receiving message from another user
  - send – sends a message to another user
- documentation in English

Drawbacks and possible extensions:

Implementation of the jobs groups, besides possibility of the process launching in the background , should also allow parallel starting of several child processes in such a way that they do not have to wait until finishing of theirs siblings.

Together with the jobs implementation all child processes of the killed process should continue their activities. Without this functionality the child processes running of the killed process or process tree restructuring is very complicated, so the entire subtree of child processes finishes.

Input and output from the console does not lock for processes with children, so that parents have to wait for finishing of children instead of eventual continuing of theirs work that does not require any input or output. Similarly it would be more suitable if inputs and outputs had process counter which would use them in such a way: when a zero number of clients had been reached, input/output would close. Current implementation works in that way input/output is closed by process that have used it for the first time and all his child process inherits this input/output (closing is only emulated for them).

Work division from among the team members:

Jiří Kučera – virtual machines manager and corresponding components, administrative arrangements

Martin Sloup – shell, terminal console, support for sending messages between users, SVN server management

Natalia Rubinova – user applications, debugging, documentation in English

Despite of the fact that the current work represents an exercise application only, its contribution to the team members experience is definitely not negligible. The most significant contribution is getting experience with threads, learning advantages and disadvantages of work with threads in Java programming language, continual knowledge deepening of operating system functioning (above all process management). Besides that each team semester work contributes to the team work ability experience.