AGH

Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie

Dokumentacja do projektu

Command line interpreter

z przedmiotu

Programowanie obiektowe

Kierunek: Elektronika II

Kacper Filipek

piątek 13:00

Prowadzący: Rafał Frączek

Kraków, 27 stycznia 2023

1. Opis projektu

Program to autorska implementacja powłoki systemowej na systemy z rodziny Unix. Implementuje funkcje wywoływania programów poprzez komenty oraz ustawianie własnego znaku zachęty.

2. Project description

This program is an original implementation of a Unix shell. It implements functions such as executing programs by a command and setting a custom command prompt.

3. Instrukcja użytkownika

Funkcje wbudowane

Program zawiera następujace funkcje wbudowane, które mozna wywołać z wiersza poleceń:

- `cd` - zmienia aktualny katalog
  Używanie: `cd path`, gdzie `path` to ścieżka do katalogu, relatywna lub absolutna
- `vars` - wyświetla zmienne lokalne dla aktualnej sesji powłoki
- `welcome` - wyświetla wiadomość powitalną z podstawowymi informacjami
- `help` - wyświetla pomoc z omówieniem najważniejszych funkcji
- `exit` - wychodzi z programu i zapisuje historię komend do pliku

Wywoływanie programów

Podstawową funckją programu jest wywoływanie programów poprzez interfejs wiersza poleceń. Aby wywołać program, należy wpisać nazwę jego pliku wykonywalnego oraz listę argumentów. Program zostanie wywołany, jeśli znajduje się w katalogu ze zmiennej PATH. Można również wywoływać programy podając ich ścieżkę absolutną lub relatywną. Przykładowe użycia:

- `ls -lah` - wywołanie ze zmiennej PATH
- `/usr/sbin/usermod -a -G sudo username` - wywołanie poprzez ścieżkę absolutną
- `./my_script.sh` - wywołanie poprzez ścieżkę relatywną

Możliwe jest przekazanie argumentów ze znakami białymi przy pomocy cudzysłowów. Ciągi znaków objęte cudzysłowem zostaną w całości przekazane do programu jako jeden argument. Przykładowo:

- `git commit -m "krotki opis commita"`
- `cat "nazwa pliku ze spacjami.txt"`

Zmienne lokalne

Program zawiera podstawowe wsparcie dla zmiennych lokalnych. Zmienną przypisuje się przy pomocy znaku równości w następujący sposów.
`NAME=VALUE`, gdzie NAME to nazwa zmiennej, a VALUE to jej wartość. Nazwa zmiennej musi spełniać następują założenia:

- Składać się z przynajmniej jednego znaku
- Zaczynać się od litery lub znaku podkreślenia (_).
- Następne znaki mogą być literami, podkreśleniami, lub cyframi

Wokół znaku równości nie mogą znajdować się żadne znaki białe. Podobnie jak w wywyływaniu możemy użyć cudzysłowów, aby przekazać większą liczbę znaków jako wartość, np: `ZMIENNA="Litwo, ojczyzno moja!"`.

Zmienne środowiskowe

Program umożliwia ustawianie zmiennych środowiskowych. Ich przypisywanie odbywa się w sposób podobny do zmiennych lokalnych, z tym że przypisanie poprzedzone jest słowem kluczowym "export". Przykładowo: `export XDG_CONFIG_HOME="/home/user/.config"`

Znak zachęty

W programie możliwe jest zdefiniowanie własnego znaku zachęty poprzez modyfikację zmiennej lokalnej `PROMPT`. Zdefiniowane są specjalne sekwencje, pozwalające wyświetlić pewne użyteczne informacje w znaku zachęty.

Lista sekwencji specjalnych

- `%u` - nazwa użytkownika
- `%h` - nazwa hosta
- `%w` - aktualny katalog

Oprócz tego możliwe jest używanie kolorów przez poniższe sekwencje. Tekst następujący po sekwencji będzie miał odpowiadający jej kolor.

- `%{red}` - czerwony
- `%{green}` - zielony
- `%{yellow}` - żółty
- `%{blue}` - niebieski
- `%{magenta}` - magneta
- `%{cyan}` - cyjan
- `%{white}` - biały
- `%{default}` - domyślny kolor tekstu w terminalu

Plik konfiguracyjny

Program posiada plik konfiguracyjny zlokalizowany po ścieżką `.config/yosh/yoshrc`, który jest interpretowany linijka po linijce na starcie programu. Można w nim zdefiniować zmienne środowiskowe i lokalne lub wywoływać programy.

4. Kompilacja

Program został napisany na systemy operacyjne z rodziny Linux i nie jest możliwe skompilowanie go na system Windows, ponieważ wymaga on funkcji z pliku nagłówkowego `unistd.h`, takich jak `fork()` lub `waitpid()`. Prawdopodobnie kompilacja programu na systemy z rodziny BSD jest możliwa, ponieważ jest on zgodny ze standardem POSIX, jednak program nie był na nich testowany.
Program można skompilować w następujący sposób.

1. Utworzyć folder `build/` poleceniem `mkdir build`
2. Wejść do folderu `build/`
3. Wykonać polecenie `cmake .. && make`

Po zbudowaniu plik wykonywalny będzie pod ścieżką `build/yosh`.

5. Pliki źródłowe

Projekt składa się z następujących plików źródłowych:

- `parser.hpp`, `parser.cpp` – deklaracja oraz implementacja klasy Parser
  - `Parser()` - konstruktor klasy
  - `Command parse(std:string command)` - funkcja parsująca polecenie w stringu na dedykowaną strukturę
  - `char** parse_to_cstrings(std::vector<std::string> args)` - funkcja konwertująca wektor stringów na tablicę c-stringów
- `prompt.hpp`, `prompt.cpp` - deklaracja oraz implementacja klasy Prompt
  - `void display()` - wypisuje znak zachęty do wyjścia standardowego
  - `void set_prompt()` - setter znaku zachęty
- `shell.hpp`, `shell.cpp` - deklaracja oraz implementacja klasy Shell
  - `unsigned int execute_command(std::string command)` - wykonuje polecenie zapisane jako string
  - `void init()` - inicjuje stan powłoki
  - `void loop()` - główna pętla powłoki
- `utils.hpp`, `utils.cpp` - deklaracje i implementacje funkcji pomocniczych
- `main.cpp` – główny plik z implementacją funkcji main.

6. Zależności

Program korzysta z następujących programów do kompilacji.

- GNU make - program do automatyzacji kompilacji programu
- CMake - generator projektów do C++

Obie powyższe są dostępne w domyślnych repozytoriach większości dystrybucji Linuxa, jak i w systemach z rodziny BSD.

7. Opis klas

W projekcie utworzono następujące klasy:

- `Parser` - klasa parsująca komendy
- `Prompt` - klasa implementująca znak zachęty
- `Shell` - klasa implementująca główne elementy powłoki

8. Zasoby

Program używa dwóch plików do działania.

- `.config/yosh/yoshrc` - plik zawierający komendy, które automatycznie są wykonywane przed startem programu.
- `.cache/yosh/history` - plik zawierający historię wyołanych poleceń. Jeśli plik nie istnieje, to program sam go stworzy i zapisze do niego historię.

9. Dalszy rozwój i ulepszenia

Program jest bardzo podstawową implementacją powłoki, więc istnieje duże pole do jego rozbudowy o dodatkową funkcjonalność. Funkcjami, do dalszej implementacji mogą być:

- przekierowywanie wyjścia programu
- przekazywanie danych między programami (pipe)
- odnoszenie się do zmiennych lokalnych
- przeszukiwanie historii
- podstawowe wsparcie dla pisania skryptów

10. Inne

Z uwagi na konieczność korzystania z API napisanego w języku C do wchodzenia w interakcji z systemem operacyjnym, program w niektórych miejscach używa dynamicznej alokacji i wskaźników z C w celu kompatybilności z funkcjami z nagłówka `unistd.h`.