**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ**

**И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»**

(СПбГУТ)

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ (**ИТПИ**)

КАФЕДРА ПРОГРАММНОЙ ИНЖЕНЕРИИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ (ПИ И ВТ)

**Лабораторная работа №1**

по дисциплине «Операционные системы и сети»

Тема «Shell-скрипты»

| Студент гр. | ИКПИ-23 |  | 07.11.2024 | Даненко Д. А. Харлова А. А. |
| --- | --- | --- | --- | --- |
| Преподаватель |  |  |  | Дагаев А. В. |

г.Санкт-Петербург

2024 г.

Оглавление

[**Цель работы** 3](#_30j0zll)

[**Постановка задачи** 3](#_1fob9te)

**Ход выполнения** [3](#_3znysh7)

[**Результаты работы**](#_3rdcrjn)4

**Вывод**4

[**Код скрипта**](#_lnxbz9) 4

# **Цель работы**

Разработать собственный командный процессор, работающий, в интерактивном режиме в текстовом окне терминала.

# **Постановка задачи**

Написать код на языке программирования C для собственной реализации разновидности командной оболочки, выполняющей следующие функции: смена текущей директории (go), выход из командной оболочки (exit), вызов списка реализованных функций (help).

# **Ход выполнения**

Реализуемую задачу было решено написать на языке программирования C ввиду его популярности, быстроты исполнения и удобства применения в данной области. Код содержит ряд функций, каждая из которых описана ниже.

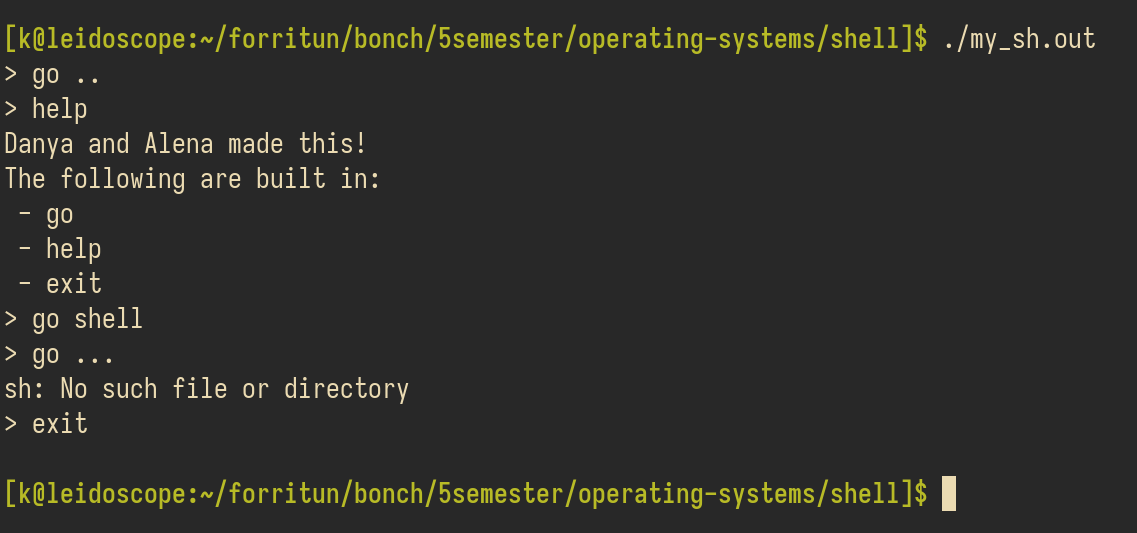
* **void loop()** - создается бесконечный цикл, который постоянно запрашивает у пользователя ввод, парсит его на аргументы и затем выполняет команду.
* **char \*read\_line(void)** - читает строку ввода от пользователя и возвращает её.
* **char \*\*split\_line(char \*line)** - используется для разбивки строки на отдельные аргументы (токены).
* **int launch(char \*\*args)** - предназначена для запуска внешней программы.
* Следующие функции являются реализациями встроенных команд командного процессора:
* **int go(char \*\*args)** - изменяет текущую директорию на ту, которая указана в качестве аргумента. Если аргумент отсутствует, выводит ошибку.
* **int sh\_help(char \*\*args)** - выводит список всех встроенных команд
* **int sh\_exit(char \*\*args)** - завершает работу командного интерпретатора.
* **int num\_builtins()** - возвращает количество встроенных команд, чтобы использовать эту информацию в других частях кода.
* **int execute(char \*\*args)** - определяет, является ли команда встроенной или внешней программой. Если команда встроенная, вызывает соответствующую функцию. Если нет, передает управление функции launch.

В программе было решено использовать следующие библиотеки:

* **<sys/wait.h> -** используется для взаимодействия с системой управления процессами. Содержит функции и макросы, необходимые для ожидания завершения дочерних процессов:
* waitpid – ожидание завершения определенного процесса.
* Макросы для анализа статуса завершившегося процесса (WIFEXITED, WIFSIGNALED и пр.).
* **<unistd.h>** - включает различные функции POSIX-систем, такие как:
* fork – создание нового процесса.
* execvp – замена текущего процесса другим.

# **Результаты работы**

На рисунке 1 показан пример работы разработанной командной оболочки.



*Рис 1 – Распределение трафика по приложениям*

**Вывод**

1. Разработан собственный командный процессор, работающий, в интерактивном режиме в текстовом окне терминала.
2. Код написан на языке программирования C .
3. Были разработаны следующие функции: смена текущей директории (go), выход из командной оболочки (exit), вызов списка реализованных функций (help).
4. Поставленная задача выполнена успешно: командная оболочка корректно работает.

# **Приложение**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/wait.h>

#include <unistd.h>

#define RL\_BUFSIZE 1024

#define TOK\_BUFSIZE 64

#define TOK\_DELIM " \t\r\n\a"

char\*

read\_line(void)

{

int bufsize = RL\_BUFSIZE;

int position = 0;

char \*buffer = malloc(sizeof(char) \* bufsize);

int c;

if (!buffer) {

fprintf(stderr, "allocation error\n");

exit(EXIT\_FAILURE);

}

while (1) {

// Read a character

c = getchar();

// If there is EOF, replace it with a null character and return

if (c == EOF || c == '\n') {

buffer[position] = '\0';

return buffer;

} else {

buffer[position] = c;

}

position++;

// If there is executon of the buffer, rellocate

if (position >= bufsize) {

bufsize += RL\_BUFSIZE;

buffer = realloc(buffer, bufsize);

if (!buffer) {

fprintf(stderr, "allocation error\n");

exit(EXIT\_FAILURE);

}

}

}

}

char\*\*

split\_line(char \*line)

{

int bufsize = TOK\_BUFSIZE;

int position = 0;

char \*\*tokens = malloc(bufsize \* sizeof(char\*));

char \*token;

if (!tokens) {

fprintf(stderr, "allocation error\n");

exit(EXIT\_FAILURE);

}

token = strtok(line, TOK\_DELIM);

while (token != NULL) {

tokens[position] = token;

position++;

if (position >= bufsize) {

bufsize += TOK\_BUFSIZE;

tokens = realloc(tokens, bufsize \* sizeof(char\*));

if (!tokens) {

fprintf(stderr, "allocation error\n");

exit(EXIT\_FAILURE);

}

}

token = strtok(NULL, TOK\_DELIM);

}

tokens[position] = NULL;

return tokens;

}

int

launch(char \*\*args)

{

pid\_t pid, wpid;

int status;

pid = fork();

if (pid == 0) {

// Child process

if (execvp(args[0], args) == -1) {

perror("sh");

}

exit(EXIT\_FAILURE);

} else if (pid < 0) {

// Error forking

perror("sh");

} else {

// Parent process

do {

wpid = waitpid(pid, &status, WUNTRACED);

} while (!WIFEXITED(status) && !WIFSIGNALED(status));

}

return 1;

}

// and sooooo onn

// Functions Declarations for builtin shell commands:

int go(char \*\*args);

int sh\_help(char \*\*args);

int sh\_exit(char \*\*args);

// List of builtin commands, followed by their corresponding functions:

char \*builtin\_str[] = {

"go",

"help",

"exit"

};

int (\*builtin\_func[]) (char \*\*) = {

&go,

&sh\_help,

&sh\_exit

};

int

num\_builtins()

{

return sizeof(builtin\_str) / sizeof(char \*);

}

// Builtin function implementations

int

go(char \*\*args)

{

if (args[1] == NULL) {

fprintf(stderr, "expected argument to \"go\"\n");

} else {

if (chdir(args[1]) != 0) {

perror("sh");

}

}

return 1;

}

int

sh\_help(char \*\*args)

{

int i;

printf("Danya and Alena made this!\n");

printf("The following are built in:\n");

for (int i = 0; i < num\_builtins(); ++i) {

printf(" - %s\n", builtin\_str[i]);

}

return 1;

}

int

sh\_exit(char \*\*args)

{

return 0;

}

int

execute(char \*\*args)

{

int i;

if (args[0] == NULL) {

// Empty command

return 1;

}

for (int i = 0; i < num\_builtins(); ++i) {

if (strcmp(args[0], builtin\_str[i]) == 0) {

return (\*builtin\_func[i])(args);

}

}

return launch(args);

}

void

loop(void)

{

char \*line;

char \*\*args;

int status;

do {

printf("> ");

line = read\_line();

args = split\_line(line);

status = execute(args);

free(line);

free(args);

} while (status);

}

int

main(int argc, char \*argv[])

{

loop();

return EXIT\_SUCCESS;

}