



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ОБРАЗОВАНИЯ «ЛИПЕЦКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Институт
Кафедра

компьютерных наук
автоматизированных систем управления

Лабораторная работа №4
по операционным системам

Студент АС-21-1

(подпись, дата)

Станиславчук С. М.

Руководитель
Доцент, к.п.н.

(подпись, дата)

Кургасов В. В.

Липецк 2024

Содержание

1. Задание кафедры.....	3
2. Ход работы.....	4
3. Тестирование.....	12
4. Вывод.....	13

1. Задание кафедры

Требуется написать простую версию командной строки, которая принимает на вход строки вида:

> command param1 param2 ...

и выполняет их при помощи вызова команды command с параметрами, то есть работает как терминал (точнее оболочка)

2. Ход работы

Напишем программу, которая будет выполнять базовые функции оболочки (shell), подобно bash. Реализуем возможность вызова и обработку базовых команд: cd, ls, grep и т.д. (вообщем те, что лежат в /bin/. Кстати python тоже там есть, а вот cd нет, но и по заданию нам все равно требуется написать собственную реализацию этой команды).

Для этого напишем программу main.c.

Для начала реализуем функцию, которая будет разделять нашу вводимую команду на токены:

```
char **parse_line(char *line) {  
    int bufsize = MAX_ARGS, position = 0;  
    char **tokens = malloc(bufsize * sizeof(char*));  
    char *token;  
  
    if (!tokens) {  
        fprintf(stderr, "allocation error\n");  
        exit(EXIT_FAILURE);  
    }  
  
    // Split the line into tokens  
    token = strtok(line, "\t\r\n\a");  
    while (token != NULL) {  
        tokens[position] = token;  
        position++;  
  
        if (position >= bufsize) {  
            bufsize += MAX_ARGS;  
            tokens = realloc(tokens, bufsize * sizeof(char*));  
            if (!tokens) {  
                free(tokens);  
                tokens = NULL;  
                break;  
            }  
        }  
        token = strtok(NULL, "\t\r\n\a");  
    }  
    tokens[position] = NULL;  
    return tokens;  
}
```

```

        fprintf(stderr, "allocation error\n");
        exit(EXIT_FAILURE);
    }

}

token = strtok(NULL, "\t\r\n\0");
}

tokens[position] = NULL;
return tokens;
}

```

А теперь нужно чтобы наши потоки выводов можно было передавать между командами, используя pipe(). То есть реализовать функционал символа « | » в bash-shell:

```

void handle_pipe(char **args) {
    int pipefd[2];
    pid_t pid1, pid2;

    pipe(pipefd);
    pid1 = fork();

    if (pid1 == 0) {
        dup2(pipefd[1], STDOUT_FILENO);
        close(pipefd[0]);
        close(pipefd[1]);

        char **cmd1 = args;
        for (int i = 0; args[i] != NULL; i++) {
            if (strcmp(args[i], "|") == 0) {
                args[i] = NULL;
                cmd1 = args;

```

```

        break;
    }
}

execute_command(cmd1);

}

pid2 = fork();

if (pid2 == 0) {
    dup2(pipefd[0], STDIN_FILENO);
    close(pipefd[1]);
    close(pipefd[0]);

    char **cmd2 = NULL;
    for (int i = 0; args[i] != NULL; i++) {
        if (strcmp(args[i], "|") == 0) {
            cmd2 = &args[i + 1];
            break;
        }
    }
    execute_command(cmd2);
}

close(pipefd[0]);
close(pipefd[1]);
waitpid(pid1, NULL, 0);
waitpid(pid2, NULL, 0);
}

```

Функция, которая будет вызывать наши команды, используя execvp:

```
void execute_command(char **args) {
```

```

if (execvp(args[0], args) == -1) {
    perror("execvp");
}
exit(EXIT_FAILURE);
}

```

Функция, которая отвечает за запись выходного потока в файл (>) и (») для добавления в конец файла, используя open().

```

int handle_redirection(char **args) {
    int fd;

    for (int i = 0; args[i] != NULL; i++) {
        if (args[i][0] == '>' && args[i+1][0] == '>') {
            // Output redirection (append)
            args[i] = NULL;
            args[i+1] = NULL;
            char *filename = args[i + 2];
            if (filename == NULL) {
                fprintf(stderr, "No output file specified for redirection.\n");
                return -1;
            }
            fd = open(filename, O_WRONLY | O_CREAT | O_APPEND, 0644);
            if (fd < 0) {
                perror("open");
                return -1;
            }
            dup2(fd, STDOUT_FILENO);
            close(fd);
            return 1;
        } else if (strcmp(args[i], ">") == 0) {

```

```

// Output redirection (overwrite)

args[i] = NULL;

char *filename = args[i + 1];

if (filename == NULL) {

    fprintf(stderr, "No output file specified for redirection.\n");

    return -1;

}

fd = open(filename, O_WRONLY | O_CREAT | O_TRUNC, 0644);

if (fd < 0) {

    perror("open");

    return -1;

}

dup2(fd, STDOUT_FILENO);

close(fd);

return 1;

}

}

return 0;
}

```

С помощью этой функции оболочка может поддерживать токены, написанные без пробелов, а также esc-код.

```

char *insert_spaces(char *line) {

    char *buffer = malloc(MAX_LINE * 2); // Allocate a larger buffer for the new line

    int j = 0;

```

```
int escape = 0; // Flag to track whether the previous character was an escape character
```

```
for (int i = 0; line[i] != '\0'; i++) {  
    if (line[i] == '>' && !escape) {  
        if (i > 0 && line[i - 1] != ' ') {  
            buffer[j++] = ' ';  
        }  
        buffer[j++] = line[i];  
        if (line[i + 1] != ' ' && line[i + 1] != '\0') {  
            buffer[j++] = ' ';  
        }  
    } else if (line[i] == '|' && !escape) {  
        if (i > 0 && line[i - 1] != ' ') {  
            buffer[j++] = ' ';  
        }  
        buffer[j++] = line[i];  
        if (line[i + 1] != ' ' && line[i + 1] != '\0') {  
            buffer[j++] = ' ';  
        }  
    } else if (line[i] == '<' && !escape) {  
        if (i > 0 && line[i - 1] != ' ') {  
            buffer[j++] = ' ';  
        }  
        buffer[j++] = line[i];  
        if (line[i + 1] != ' ' && line[i + 1] != '\0') {  
            buffer[j++] = ' ';  
        }  
    } else if (line[i] == '\"' && !escape) {  
        // Ignore double quotes  
        continue;  
    } else if (line[i] == '\\') {  
        if (line[i + 1] == '\\') {
```

```

        buffer[j++] = line[i];

    }

    escape = !escape; // Toggle escape flag

} else {

    buffer[j++] = line[i];

    escape = 0; // Reset escape flag

}

buffer[j] = '\0';

return buffer;

}

```

Функция, реализующая навигацию по директориям:

```

void change_directory(char *path) {
    if (chdir(path) != 0) {
        perror("chdir");
    }
}

```

Остается лишь в функции main() запустить цикл, ожидающий ввода команд:

```

int main() {
    char line[MAX_LINE];
    char *args[MAX_ARGS];
    int should_run = 1;

    printf("This is my own shell\n");
    while (should_run) {

```

```

printf("> ");
fflush(stdout);

if (!fgets(line, MAX_LINE, stdin)) {
    break;
}

// Remove comments
char *comment = strchr(line, '#');
if (comment) {
    *comment = '\0';
}

// Insert spaces around special characters
char *processed_line = insert_spaces(line);

// Parse the line
char **args = parse_line(processed_line);

// Handle the built-in cd command
if (args[0] && strcmp(args[0], "cd") == 0) {
    if (args[1]) {
        change_directory(args[1]);
    } else {
        fprintf(stderr, "cd: missing argument\n");
    }
    free(processed_line);
    continue;
}

// Create a new process
pid_t pid = fork();

```

```

if (pid == 0) {
    // Check for a pipeline
    for (int i = 0; args[i] != NULL; i++) {
        if (strcmp(args[i], "|") == 0) {
            handle_pipe(args);
            exit(EXIT_SUCCESS);
        }
    }

    // Handle redirection
    handle_redirection(args);

    // Execute the command
    execute_command(args);
} else if (pid < 0) {
    perror("fork");
} else {
    waitpid(pid, NULL, 0);
}

free(processed_line);
}

return 0;
}

```

3. Тестирование

А теперь протестируем shell:

```

alacritty
[stanik@archlinux ~]$ ./programmer/++Programmer/3 курс 2/Операционные системы/lr4/cprog$ ./cmd
This is my own shell
> ps aux | grep init
root      1  0.0  0.0 22260 13348 ?        Ss   10:25  0:01 /sbin/init
stanik    55962 0.0  0.0 6576 2432 pts/0  S+  13:53  0:00 grep init
> echo "first data" > test.txt
> echo "second data" >> test.txt
> cat test.txt
first data
second data
> python
Python 3.11.8 (main, Feb 12 2024, 14:50:05) [GCC 13.2.1 20230801] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import math
>>> math.log(8,2)
3.0
>>> exit()
> echo "print('result is ', 123 + 456)" | python -i | grep result
python: can't open file '/home/stanik/programmer/++Programmer/3 курс 2/Операционные системы/lr4/cprog/': [Errno 2] No such file or directory
>>> result is 579
>>>
> ls
>' cmd  file  file.txt  main.c  test.txt  tetx.txt  text.txt
> cd ..
> ls
cprog  tests
> cd cprog
> ls
'>' cmd  file  file.txt  main.c  test.txt  tetx.txt  text.txt
> []

```

```

[stanik@archlinux ~]$ ps aux | grep init
root      1  0.0  0.0 22260 13348 ?        Ss   10:25  0:01 /sbin/init
stanik    55976 0.0  0.0 6572 2560 pts/4  S+  13:53  0:00 grep --color=auto init
[stanik@archlinux ~]$ echo "first data" > test.txt
[stanik@archlinux ~]$ echo "second data" >> test.txt
[stanik@archlinux ~]$ cat test.txt
first data
second data
[stanik@archlinux ~]$ python
Python 3.11.8 (main, Feb 12 2024, 14:50:05) [GCC 13.2.1 20230801] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import math
>>> math.log(8,2)
3.0
>>> exit()
[stanik@archlinux ~]$ echo "print('result is ', 123 + 456)" | python -i | grep result
Python 3.11.8 (main, Feb 12 2024, 14:50:05) [GCC 13.2.1 20230801] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> result is 579
[stanik@archlinux ~]$ ls
blenderModels  Downloads  Pictures  scripts  UnityProj ]  sudo
c_proj          hdpSample  plugins   temp     file.txt  test.txt
Desktop         ly         programmer  Templates  vimsource  goboz_ip_ranges.txt
Documents       Minecraft  Public    temrite   windows  idz_05
Downloads       music     screenshots  Unity     yag-git   math
[stanik@archlinux ~]$ cd ..
[stanik@archlinux home]$ ls
[stanik@archlinux home]$ 

```

Рисунок 1 — сравнение поведения моей оболочки и оболочки Bash

```

[stanik@archlinux ~]$ ./programmer/++Programmer/3 курс 2/Операционные системы/lr4/cprog$ ./cmd
This is my own shell
> echo "some text\" with quote" > test.txt
> cat test.txt
some text" with quote
> []

```

```

[stanik@archlinux ~]$ echo "some text\" with quote" > test.txt
[stanik@archlinux ~]$ cat test.txt
some text" with quote
[stanik@archlinux ~]$ 

```

Рисунок 2 — сравнение поведения моей оболочки и оболочки Bash на примере учитывания ковычек после esc-кода

Как можно заметить, результаты идентичны.

4. Вывод

В ходе выполненной работы написал свою упрощенную bash-подобную оболочку, используя язык программирования C, согласно всем дополнительным условиям задания.