**Лабораторная работа №2,** «Создание простой командной оболочки»  
Группа: Б23-534  
ФИО: Калашников Владимир Алексеевич  
Номер в журнале: 5  
Год в Москве: 2025

**Описание лабораторной работы**

[Ссылка на GitHub](https://github.com/SD5th/Informatika_Labs_MIFI/tree/main/4Sem/LW_4sem-2)

Работа выполнена на языке Си с использованием и включает в себя два файла:   
1) *ProstoShell.h*, содержит в себе реализацию командной оболочки   
2) *main.c*, запускает основную функцию оболочки

Также проект оснащён небольшим скриптиком *BuildAndRun.sh* для сборки с помощью CMake и последующего запуска оболочки.

**Описание кода**

В *main.c* действительно нет ничего примечательного, просто запуск оболочки

main.c  
#include "Source/ProstoShell.h"

int main() {

  shell\_loop();

  return 0;

}

Файл *ProstoShell.h*, в его начале объявление констант и подключение необходимых библиотек.

ProstoShell.h  
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/wait.h>

#include <sys/types.h>

#include <dirent.h>

#define MAX\_LEN 1024

#define MAX\_ARGS 64

Объявление всех функций в начале файла

void shell\_loop();

void parse\_args(char \*line, char \*\*args);

void execute\_command(char \*\*args);

void execute\_pipeline(char \*\*args1, char \*\*args2);

int is\_builtin(char\* cmd);

void handle\_builtin(char \*\*args);

void bgrep(char \*pattern);

Встроенная функция *bgrep*, отсеивающая из потока ввода лишь те элементы, что содержат строку pattern.

void bgrep(char \*pattern) {

  char line[MAX\_LEN];

  if (pattern == NULL || pattern[0] == '\0') {

    fprintf(stderr, "bgrep: missing pattern\n");

    return;

  }

  size\_t pattern\_len = strlen(pattern);

  while (fgets(line, MAX\_LEN, stdin)) {

    if (strstr(line, pattern) != NULL) {

      printf("%s", line);

    }

  }

}

Функции *is\_builtin* и *handle\_builtin* отвечают за исполнение встроенных команд. Используются в *execute\_command* и *execute\_pipeline*, если функция не является встроенной, то она будет запущена силами *execvp*.

int is\_builtin(char\* cmd) {

  return  strcmp(cmd, "cd") == 0 ||

          strcmp(cmd, "exit") == 0 ||

          strcmp(cmd, "bgrep") == 0;

}

void handle\_builtin(char \*\*args) {

  if (strcmp(args[0], "cd") == 0) {

    if (args[1] == NULL) {

      fprintf(stderr, "cd: missing argument\n");

    }

    else {

      if (chdir(args[1]) != 0) {

        perror("cd");

      }

    }

  }

  else if (strcmp(args[0], "exit") == 0) {

    exit(0);

  }

  else if (strcmp(args[0], "bgrep") == 0) {

    if (args[1] == NULL) {

      fprintf(stderr, "bgrep: missing pattern\n");

    }

    else {

      bgrep(args[1]);

    }

  }

}

Разбивает строку на аргументы.

void parse\_args(char \*line, char \*\*args) {

  int i = 0;

  char\* token = strtok(line, " \t\n");

  while (token != NULL && i < MAX\_ARGS - 1) {

    args[i++] = token;

    token = strtok(NULL, " \t\n");

  }

  args[i] = NULL;

}

Функция *execute\_command* проверяет, есть ли команда среди встроенных, затем или перенаправляет исполнение к *handle\_builtin*, или создаёт *fork*, где команда выполняется силами *execvp*.

void execute\_command(char \*\*args) {

  if (args[0] == NULL) {

    return;

  }

  if (is\_builtin(args[0])) {

    handle\_builtin(args);

    return;

  }

  pid\_t pid = fork();

  if (pid == 0) {

    execvp(args[0], args);

    perror("execvp");

    exit(1);

  } else if (pid < 0) {

    perror("fork");

  } else {

    wait(NULL);

  }

}

*execute\_pipeline* реализовывает выполнение двух команд через **“pipe”**.   
Сперва создаёт контейнер (тоже pipe), через который смогут общаться два процесса. Затем создаёт сами процессы для левой и правой команды, соединяет вывод левого со вводом контейнера, ввод правого с выводом контейнера, запускает процессы и ждёт конца работы. Как и *execute\_command*, умеет отделять и исполнять встроенные команды.

void execute\_pipeline(char \*\*args1, char \*\*args2) {

  int pipefd[2];

  pid\_t pid1, pid2;

  if (pipe(pipefd)) {

    perror("pipe");

    return;

  }

  pid1 = fork();

  if (pid1 == 0) {

    close(pipefd[0]);

    dup2(pipefd[1], STDOUT\_FILENO);

    close(pipefd[1]);

    if (is\_builtin(args1[0]))

    {

      handle\_builtin(args1);

      exit(EXIT\_SUCCESS);

    }

    else

    {

      execvp(args1[0], args1);

      perror("execvp");

      exit(EXIT\_FAILURE);

    }

  }

  pid2 = fork();

  if (pid2 == 0) {

    close(pipefd[1]);

    dup2(pipefd[0], STDIN\_FILENO);

    close(pipefd[0]);

    if (is\_builtin(args2[0]))

    {

      handle\_builtin(args2);

      exit(EXIT\_SUCCESS);

    }

    else

    {

      execvp(args2[0], args2);

      perror("execvp");

      exit(EXIT\_FAILURE);

    }

  }

  close(pipefd[0]);

  close(pipefd[1]);

  waitpid(pid1, NULL, 0);

  waitpid(pid2, NULL, 0);

}