Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации Сибирский Государственный Университет Телекоммуникаций и Информатики СибГУТИ

Кафедра вычислительных систем

**Лабораторная работа №2**

по дисциплине «Операционные системы»

Выполнил:

Студент группы ИВ-921

Ярошев Р.А.

Работу проверил:

ассистент кафедры ВС

Петрук Е. А.

Новосибирск 2021

## Задание

Реализовать упрощенную командную оболочку (аналог bash, csh, и т.д.). Обо-

лочка должна предоставлять следующий функционал:

• Выводить приглашение для ввода команды;

• Запускать введенную команду, выводить результат;

• Поддерживать конвейер:

> command1 | command2 | command3

• Любой другой функционал на ваше усмотрение.

Системные вызовы, которые понадобятся вам при выполнении лаборатор-

ной (прочитать о них в мануале): fork, exec, wait, exit, waitpid, getpid,

pipe, dup2.

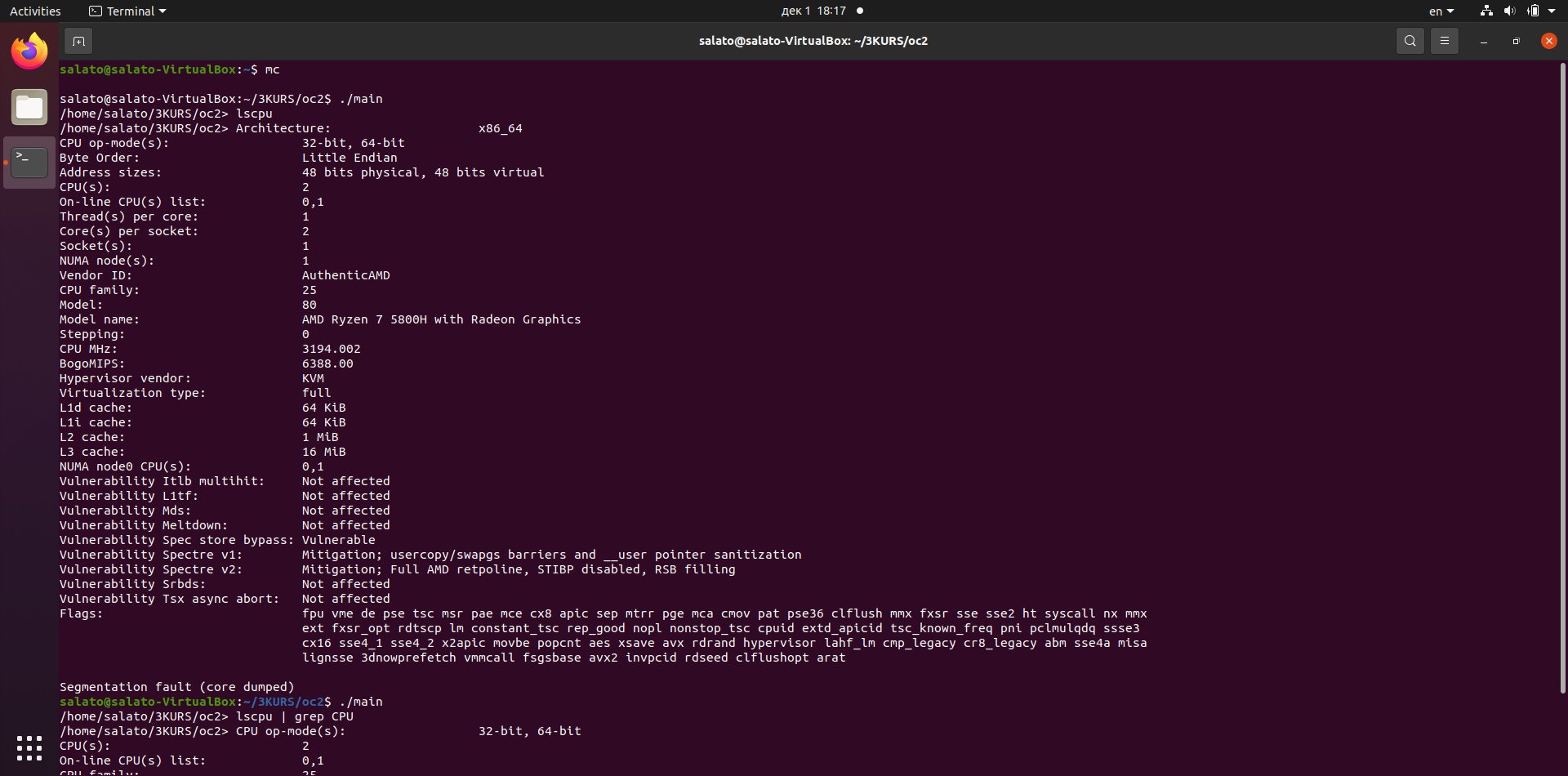
## Результат работы программы

Сборка:

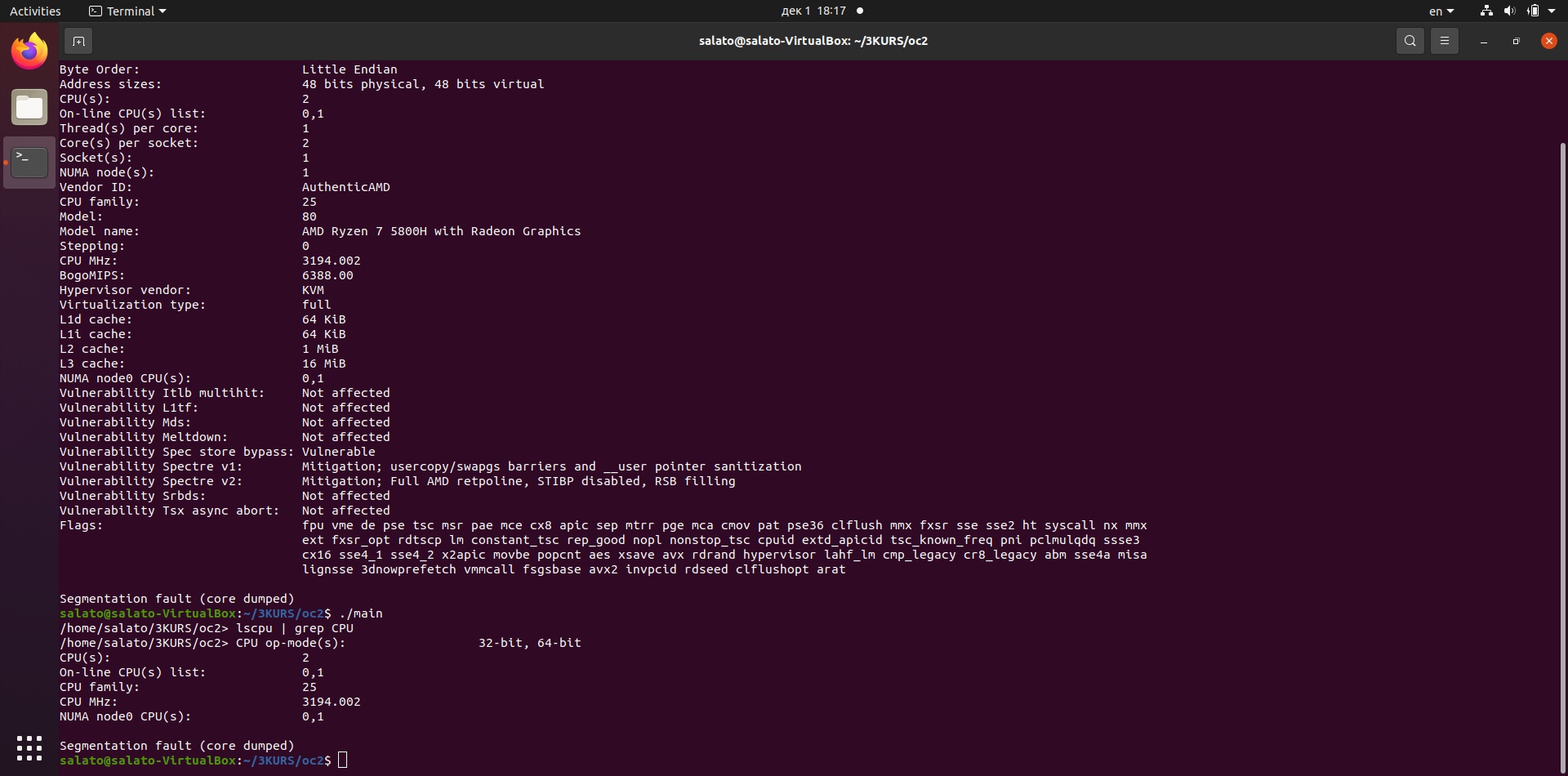
$ gcc main.c

Запуск:

$ ./main.out



Изображение 1. Сборка приложения и запуск программы



Изображение 2. Проверка работы конвейера

## Листинг

main.c

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <errno.h>

#define MAX\_LINE\_LENGTH 1024

#define BUFFER\_SIZE 64

#define REDIR\_SIZE 2

#define PIPE\_SIZE 3

#define MAX\_COMMAND\_NAME\_LENGTH 128

#define PROMPT\_MAX\_LENGTH 30

#define PIPE\_OPT "|"

int running = 1;

char \*get\_current\_dir() {

char cwd[FILENAME\_MAX];

char\*result = getcwd(cwd, sizeof(cwd));

return result;

}

char \*prompt() {

static char \*\_prompt = NULL;

if (\_prompt == NULL) {

\_prompt = malloc(PROMPT\_MAX\_LENGTH \* sizeof(char));

if (\_prompt == NULL) {

perror("Error: Unable to locate memory");

exit(EXIT\_FAILURE);

}

}

}

void error\_alert(char \*msg) {

printf("%s %s**\n**", prompt(), msg);

}

void remove\_end\_of\_line(char \*line) {

int i = 0;

while (line[i] != '**\n**') {

i++;

}

line[i] = '**\0**';

}

void read\_line(char \*line) {

char \*ret = fgets(line, MAX\_LINE\_LENGTH, stdin);

remove\_end\_of\_line(line);

if (strcmp(line, "exit") == 0 || ret == NULL || strcmp(line, "quit") == 0) {

exit(EXIT\_SUCCESS);

}

}

void parse\_command(char \*input\_string, char \*\*argv, int \*wait) {

int i = 0;

while (i < BUFFER\_SIZE) {

argv[i] = NULL;

i++;

}

\*wait = (input\_string[strlen(input\_string) - 1] == '&') ? 0 : 1;

input\_string[strlen(input\_string) - 1] = (\*wait == 0) ? input\_string[strlen(input\_string) - 1] = '**\0**'

: input\_string[strlen(input\_string) - 1];

i = 0;

argv[i] = strtok(input\_string, " ");

if (argv[i] == NULL) return;

while (argv[i] != NULL) {

i++;

argv[i] = strtok(NULL, " ");

}

argv[i] = NULL;

}

int is\_pipe(char \*\*argv) {

int i = 0;

while (argv[i] != NULL) {

if (strcmp(argv[i], PIPE\_OPT) == 0) {

return i;

}

i = -~i;

}

return 0;

}

void parse\_redirect(char \*\*argv, char \*\*redirect\_argv, int redirect\_index) {

redirect\_argv[0] = strdup(argv[redirect\_index]);

redirect\_argv[1] = strdup(argv[redirect\_index + 1]);

argv[redirect\_index] = NULL;

argv[redirect\_index + 1] = NULL;

}

void parse\_pipe(char \*\*argv, char \*\*child01\_argv, char \*\*child02\_argv, int pipe\_index) {

int i = 0;

for (i = 0; i < pipe\_index; i++) {

child01\_argv[i] = strdup(argv[i]);

}

child01\_argv[i++] = NULL;

while (argv[i] != NULL) {

child02\_argv[i - pipe\_index - 1] = strdup(argv[i]);

i++;

}

child02\_argv[i - pipe\_index - 1] = NULL;

}

void exec\_child(char \*\*argv) {

if (execvp(argv[0], argv) < 0) {

fprintf(stderr, "Error: Failed to execte command.**\n**");

exit(EXIT\_FAILURE);

}

}

void exec\_child\_pipe(char \*\*argv\_in, char \*\*argv\_out) {

int fd[2];

if (pipe(fd) == -1) {

perror("Error: Pipe failed");

exit(EXIT\_FAILURE);

}

if (fork() == 0) {

dup2(fd[1], STDOUT\_FILENO);

close(fd[0]);

close(fd[1]);

exec\_child(argv\_in);

exit(EXIT\_SUCCESS);

}

if (fork() == 0) {

dup2(fd[0], STDIN\_FILENO);

close(fd[1]);

close(fd[0]);

exec\_child(argv\_out);

exit(EXIT\_SUCCESS);

}

close(fd[0]);

close(fd[1]);

wait(0);

wait(0);

}

int simple\_shell\_cd(char \*\*args);

int simple\_shell\_exit(char \*\*args);

void exec\_command(char \*\*args, char \*\*redir\_argv, int wait, int res);

char \*builtin\_str[] = {

"cd",

"exit"

};

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

&simple\_shell\_cd,

&simple\_shell\_exit

};

int simple\_shell\_num\_builtins() {

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

}

int simple\_shell\_cd(char \*\*argv) {

if (argv[1] == NULL) {

fprintf(stderr, "Error: Expected argument to **\"**cd**\"\n**");

} else {

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

perror("Error: Error when change the process's working directory to PATH.");

}

}

return 1;

}

int simple\_shell\_exit(char \*\*args) {

running = 0;

return running;

}

int simple\_shell\_pipe(char \*\*args) {

int pipe\_op\_index = is\_pipe(args);

if (pipe\_op\_index != 0) {

char \*child01\_arg[PIPE\_SIZE];

char \*child02\_arg[PIPE\_SIZE];

parse\_pipe(args, child01\_arg, child02\_arg, pipe\_op\_index);

exec\_child\_pipe(child01\_arg, child02\_arg);

return 1;

}

return 0;

}

void exec\_command(char \*\*args, char \*\*redir\_argv, int wait, int res) {

for (int i = 0; i < simple\_shell\_num\_builtins(); i++) {

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

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

res = 1;

}

}

if (res == 0) {

int status;

pid\_t pid = fork();

if (pid == 0) {

if (res == 0) res = simple\_shell\_pipe(args);

if (res == 0) execvp(args[0], args);

exit(EXIT\_SUCCESS);

} else if (pid < 0) {

perror("Error: Error forking");

exit(EXIT\_FAILURE);

} else {

if (wait == 1) {

waitpid(pid, &status, WUNTRACED);

}

}

}

}

int main(void) {

char \*args[BUFFER\_SIZE];

char line[MAX\_LINE\_LENGTH];

char t\_line[MAX\_LINE\_LENGTH];

char \*redir\_argv[REDIR\_SIZE];

int wait;

int res = 0;

while (running) {

printf("%s%s> ", prompt(), get\_current\_dir());

fflush(stdout);

read\_line(line);

parse\_command(line, args, &wait);

exec\_command(args, redir\_argv, wait, res);

}

return 0;

}