Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №1 по курсу

«Операционные системы»

Группа: М8О-213Б-23

Студент: Савинов Н. О.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 28.12.24

Москва, 2024

Постановка задачи

Вариант 14.

Родительский процесс создает два дочерних процесса. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Child1 и Child2 можно «соединить» между собой дополнительным каналом. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child1 и child2 производят работу над строками. Child2 пересылает результат своей работы родительскому процессу. Родительский процесс полученный результат выводит в стандартный поток вывода. Child1 переводит строки в нижний регистр. убирает все задвоенные пробелы.

Общий метод и алгоритм решения

Использованные системные вызовы:

● pid_t fork(void) — используется для создания дочернего процесса.

● int pipe(int fd) — создает канал для однонаправленной связи между процессами. fd[0] используется для чтения из канала, а fd[1] — для записи в него.

● ssize_t write(int fd, const void buf, size_t count) — записывает данные из буфера buf в файл, связанный с файловым дескриптором fd, в количестве байтов, указанном в count.

● ssize_t read(int fd, void buf, size_t count) — читает данные из файла или канала, связанного с файловым дескриптором fd, в буфер buf в количестве байтов, указанном в count.

● int execv(const char path, char const argv[]) — заменяет текущий процесс новым процессом, запускающим указанную программу.

● int32_t open(const char*file, int oflag, …); – открывает файл и возвращает файловый дескриптор.

● int close(int fd) – закрывает файл.

● int dup2(int oldfd, int newfd) — дублирует файловый дескриптор oldfd, заменяя им дескриптор newfd. Перенаправление стандартного ввода дочернего процесса на канал.

● int wait(int status) — приостанавливает выполнение родительского процесса до завершения дочернего процесса.

Во время выполнения лабораторной работы я разрабатывал программу, в которой родительский процесс создает два дочерних процесса для обработки строк, получаемых от пользователя. Сначала я организовал механизм передачи данных между процессами, используя каналы (pipes), что позволило мне отправлять введенные строки в первый дочерний процесс, а затем получать обработанные данные от второго дочернего процесса. Я создал два отдельных исполняемых файла для дочерних процессов: первый отвечает за преобразование строк в нижний регистр, а второй заменяет пробелы на символы подчеркивания. В родительском процессе я использовал функции fork() и execv() для создания и запуска дочерних процессов, а также перенаправил стандартные потоки ввода-вывода с помощью dup2(), чтобы установить каналы между процессами. В процессе разработки я учел обработку ошибок: проверял результаты вызовов функций, таких как pipe(), fork() и execv(), и выводил соответствующие сообщения об ошибках в

стандартный поток ошибок. Я реализовал цикл, в котором родительский процесс считывал ввод от пользователя, отправлял его через pipe1, а затем ожидал результаты от второго дочернего процесса через pipe3. После получения результатов я выводил их на экран.

Код программы

server.c

```
#include <stdint.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <unistd.h>
#include <string.h>

static char CHILD1_PROGRAM_NAME[] = "./child1";
static char CHILD2_PROGRAM_NAME[] = "./child2";

int main(int argc, char **argv) {
    if (argc != 1) {
        char msg[] =  "usage: ./{filename}\n";
        write(STDOUT_FILENO, msg, strlen(msg));
        exit(EXIT_SUCCESS);
    }

    // Get full path to the directory, where program resides
    char progpath[1024];
    {
        // Read full program path, including its name
        ssize_t len = readlink("/proc/self/exe", progpath,
                               sizeof(progpath) - 1);
        if (len == -1) {
            const char msg[] = "error: failed to read full program path\n";
            write(STDERR_FILENO, msg, sizeof(msg));
            exit(EXIT_FAILURE);
        }
```

```
    // Trim the path to first slash from the end
    while (progpath[len] != '/')
        --len;

    progpath[len + 1] = '\0';
}

// Open pipe
int pipe1[2], pipe2[2], pipe3[2];
if (pipe(pipe1) == -1 || pipe(pipe2) == -1 || pipe(pipe3) == -1) {
    const char msg[] = "error: failed to create pipe\n";
    write(STDERR_FILENO, msg, sizeof(msg));
    exit(EXIT_FAILURE);
}

const pid_t child1 = fork();

switch (child1) {
    case -1: {
        const char msg[] = "error: failed to spawn new process\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    } break;

    case 0: {
        dup2(pipe1[STDIN_FILENO], STDIN_FILENO);
        dup2(pipe2[STDOUT_FILENO], STDOUT_FILENO);

        close(pipe1[STDOUT_FILENO]);
        close(pipe2[STDIN_FILENO]);
```

```
        close(pipe3[STDIN_FILENO]);
        close(pipe3[STDOUT_FILENO]);

        {
            char *const args[] = {CHILD1_PROGRAM_NAME, NULL};

            int32_t status = execv(CHILD1_PROGRAM_NAME, args);

            if (status == -1) {
                const char msg[] = "error: failed to exec into new exectuable image\n";
                write(STDERR_FILENO, msg, sizeof(msg));
                exit(EXIT_FAILURE);
            }
        }
    } break;
}

const pid_t child2 = fork();
switch (child2) {
    case -1: {
        const char msg[] = "error: failed to spawn new process\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    } break;

    case 0: {

        dup2(pipe2[STDIN_FILENO], STDIN_FILENO);
        dup2(pipe3[STDOUT_FILENO], STDOUT_FILENO);
```

```
        close(pipe1[STDIN_FILENO]);
        close(pipe1[STDOUT_FILENO]);

        close(pipe2[STDOUT_FILENO]);
        close(pipe3[STDIN_FILENO]);

        {
            char *const args[] = {CHILD2_PROGRAM_NAME, NULL};

            int32_t status = execv(CHILD2_PROGRAM_NAME, args);

            if (status == -1) {
                const char msg[] = "error: failed to exec into new exectuable image\n";
                write(STDERR_FILENO, msg, sizeof(msg));
                exit(EXIT_FAILURE);
            }
        }
    } break;
}
// closing useless
close(pipe1[0]);
close(pipe2[0]);
close(pipe3[1]);

ssize_t bytes;
char buf[1024];

char msg_of_hint[] = "Введите строку (или пустую строку для выхода): ";
int len_of_msg_of_hint = strlen(msg_of_hint);
write(STDOUT_FILENO, msg_of_hint, len_of_msg_of_hint);
```

```
while (bytes = read(STDIN_FILENO, buf, sizeof(buf))) {
    if (bytes < 0) {
        const char msg[] = "error: failed to read from stdin\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    } else if (buf[0] == '\n') {
        break;
    }
    buf[bytes] = '\0';
    // Write into pipe1 for child1 input
    write(pipe1[1], buf, strlen(buf));

    // read from pipe3
    char result[1024];
    ssize_t bytes_read = read(pipe3[0], result, sizeof(result) - 1);
    if (bytes_read > 0) {
        result[bytes_read] = '\0';
        char msg[] = "Processed result: ";
        write(STDOUT_FILENO, msg, strlen(msg));
        write(STDOUT_FILENO, result, bytes_read - 1);
        write(STDOUT_FILENO, "\n\n", 2);
        write(STDOUT_FILENO, msg_of_hint, len_of_msg_of_hint);
    }

}
close(pipe1[1]);
close(pipe3[0]);
close(pipe2[1]);

wait(NULL);
```

```
    wait(NULL);
    return 0;
}
```

child1.c

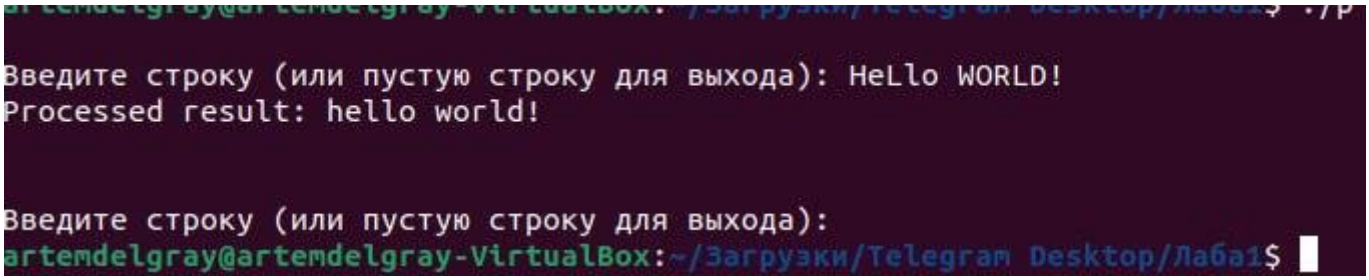
```
#include <ctype.h>
#include <unistd.h>
#include <string.h>

int main() {
    char input[1024];
    ssize_t bytes_read;

    while ((bytes_read = read(STDIN_FILENO, input, sizeof(input))) > 0) {
        input[bytes_read] = '\0';

        for (int i = 0; i < bytes_read; i++) {
            input[i] = tolower(input[i]);
        }
        write(STDOUT_FILENO, input, bytes_read);
    }
    return 0;
}
```

child2.c

```
#include <unistd.h>
#include <string.h>

int main() {
    char input[1024];
    ssize_t bytes_read;
    char output[1024];

    while ((bytes_read = read(STDIN_FILENO, input, sizeof(input))) > 0) {
        ssize_t output_index = 0;
        int space_flag = 0;

        for (ssize_t i = 0; i < bytes_read; i++) {
            if (input[i] == ' ') {
                if (!space_flag) {
                    output[output_index++] = ' ';
                    space_flag = 1;
                }
            } else {
                output[output_index++] = input[i];
                space_flag = 0;
            }
        }

        write(STDOUT_FILENO, output, output_index);
    }

    return 0;
}
```

Протокол работы программы

Тестирование:



Strace:

```
strace -f ./p

execve("./p", ["./p"], 0x7ffed156c548 /* 46 vars */) = 0

brk(NULL)                               = 0x5de87d690000

arch_prctl(0x3001 /* ARCH_??? */, 0x7ffd41dde740) = -1 EINVAL (Недопустимый аргумент)

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x717588217000

access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (Нет такого файла или каталога)

openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=58047, ...}, AT_EMPTY_PATH) = 0

mmap(NULL, 58047, PROT_READ, MAP_PRIVATE, 3, 0) = 0x717588208000

close(3)                                = 0

openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\0\0\0\0\0"..., 832) = 832

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) =
784

pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48, 848) =
48

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0I\17\357\204\3$\f\221\2039x\324\224\323\236S"...,
68, 896) = 68

newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, AT_EMPTY_PATH) = 0

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) =
784

mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x717587e00000

mprotect(0x717587e28000, 2023424, PROT_NONE) = 0

mmap(0x717587e28000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x28000) = 0x717587e28000

mmap(0x717587fbd000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) =
0x717587fbd000

mmap(0x717588016000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x215000) = 0x717588016000

mmap(0x71758801c000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1,
0) = 0x71758801c000

close(3)                                = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x717588205000

arch_prctl(ARCH_SET_FS, 0x717588205740) = 0

set_tid_address(0x717588205a10)         = 3415
```

```
set_robust_list(0x717588205a20, 24)     = 0

rseq(0x7175882060e0, 0x20, 0, 0x53053053) = 0

mprotect(0x717588016000, 16384, PROT_READ) = 0

mprotect(0x5de87c185000, 4096, PROT_READ) = 0

mprotect(0x717588251000, 8192, PROT_READ) = 0

prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0

munmap(0x717588208000, 58047)           = 0

readlink("/proc/self/exe",
"/home/artemdelgray/\320\227\320\260\320\263\321\200\321\203\320\267\320"..., 1023) = 54

pipe2([3, 4], 0)                        = 0

pipe2([5, 6], 0)                        = 0

pipe2([7, 8], 0)                        = 0

clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLDstrace: Process
3416 attached

, child_tidptr=0x717588205a10) = 3416

[pid  3415] clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD
<unfinished ...>

[pid  3416] set_robust_list(0x717588205a20, 24) = 0

[pid  3416] dup2(3, 0)                  = 0

[pid  3416] dup2(6, 1)                  = 1

[pid  3416] close(4)                    = 0

[pid  3416] close(5)                    = 0

strace: Process 3417 attached

[pid  3416] close(7)                    = 0

[pid  3416] close(8 <unfinished ...>

[pid  3417] set_robust_list(0x717588205a20, 24 <unfinished ...>

[pid  3416] <... close resumed>)        = 0

[pid  3416] execve("./child1", ["./child1"], 0x7ffd41dde918 /* 46 vars */ <unfinished ...>

[pid  3417] <... set_robust_list resumed>) = 0

[pid  3415] <... clone resumed>, child_tidptr=0x717588205a10) = 3417

[pid  3417] dup2(5, 0 <unfinished ...>

[pid  3415] close(3 <unfinished ...>

[pid  3417] <... dup2 resumed>)         = 0

[pid  3417] dup2(8, 1 <unfinished ...>
```

```
[pid  3415] <... close resumed>)        = 0

[pid  3417] <... dup2 resumed>)         = 1

[pid  3415] close(5 <unfinished ...>

[pid  3417] close(3 <unfinished ...>

[pid  3415] <... close resumed>)        = 0

[pid  3417] <... close resumed>)        = 0

[pid  3415] close(8 <unfinished ...>

[pid  3417] close(4 <unfinished ...>

[pid  3415] <... close resumed>)        = 0

[pid  3417] <... close resumed>)        = 0

[pid  3415] write(1, "Enter your string or (Enter / CT"..., 51 <unfinished ...>

[pid  3417] close(6)                    = 0

Enter your string or (Enter / CTRL + D) for stop:

[pid  3415] <... write resumed>)        = 51

[pid  3417] close(7)                    = 0

[pid  3415] read(0,  <unfinished ...>

[pid  3417] execve("./child2", ["./child2"], 0x7ffd41dde918 /* 46 vars */ <unfinished ...>

[pid  3416] <... execve resumed>)       = 0

[pid  3416] brk(NULL)                   = 0x6208235d5000

[pid  3417] <... execve resumed>)       = 0

[pid  3417] brk(NULL)                   = 0x55b49f9db000

[pid  3417] arch_prctl(0x3001 /* ARCH_??? */, 0x7ffe7878b0f0 <unfinished ...>

[pid  3416] arch_prctl(0x3001 /* ARCH_??? */, 0x7ffd70987ae0 <unfinished ...>

[pid  3417] <... arch_prctl resumed>)   = -1 EINVAL (Недопустимый аргумент)

[pid  3417] mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7c0299d7f000

[pid  3417] access("/etc/ld.so.preload", R_OK) = -1 ENOENT (Нет такого файла или каталога)

[pid  3417] openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

[pid  3417] newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=58047, ...}, AT_EMPTY_PATH) = 0

[pid  3417] mmap(NULL, 58047, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7c0299d70000

[pid  3417] close(3)                    = 0

[pid  3417] openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3

[pid  3417] read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\0\0\0\0\0"...,
832) = 832
```

```
[pid  3417] pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

[pid  3417] pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48, 848) = 48

[pid  3417] pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0I\17\357\204\3$\f\221\2039x\324\224\323\236S"..., 68, 896) = 68

[pid  3417] newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, AT_EMPTY_PATH) = 0

[pid  3417] pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

[pid  3417] mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7c0299a00000

[pid  3417] mprotect(0x7c0299a28000, 2023424, PROT_NONE) = 0

[pid  3417] mmap(0x7c0299a28000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7c0299a28000

[pid  3417] mmap(0x7c0299bbd000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) = 0x7c0299bbd000

[pid  3417] mmap(0x7c0299c16000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x7c0299c16000

[pid  3417] mmap(0x7c0299c1c000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7c0299c1c000

[pid  3417] close(3)                    = 0

[pid  3417] mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7c0299d6d000

[pid  3417] arch_prctl(ARCH_SET_FS, 0x7c0299d6d740) = 0

[pid  3417] set_tid_address(0x7c0299d6da10) = 3417

[pid  3417] set_robust_list(0x7c0299d6da20, 24) = 0

[pid  3417] rseq(0x7c0299d6e0e0, 0x20, 0, 0x53053053) = 0

[pid  3417] mprotect(0x7c0299c16000, 16384, PROT_READ) = 0

[pid  3417] mprotect(0x55b49f7d7000, 4096, PROT_READ) = 0

[pid  3417] mprotect(0x7c0299db9000, 8192, PROT_READ <unfinished ...>

[pid  3416] <... arch_prctl resumed>)   = -1 EINVAL (Недопустимый аргумент)

[pid  3417] <... mprotect resumed>)     = 0

[pid  3416] mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0 <unfinished ...>

[pid  3417] prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0

[pid  3416] <... mmap resumed>)         = 0x72f6fc24b000
```

```
[pid  3417] munmap(0x7c0299d70000, 58047 <unfinished ...>

[pid  3416] access("/etc/ld.so.preload", R_OK <unfinished ...>

[pid  3417] <... munmap resumed>)       = 0

[pid  3416] <... access resumed>)       = -1 ENOENT (Нет такого файла или каталога)

[pid  3416] openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 4

[pid  3417] read(0,  <unfinished ...>

[pid  3416] newfstatat(4, "", {st_mode=S_IFREG|0644, st_size=58047, ...}, AT_EMPTY_PATH) = 0

[pid  3416] mmap(NULL, 58047, PROT_READ, MAP_PRIVATE, 4, 0) = 0x72f6fc23c000

[pid  3416] close(4)                    = 0

[pid  3416] openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 4

[pid  3416] read(4, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\0\0\0\0\0"...,
832) = 832

[pid  3416] pread64(4, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"...,
784, 64) = 784

[pid  3416] pread64(4, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"...,
48, 848) = 48

[pid  3416] pread64(4,
"\4\0\0\0\24\0\0\0\3\0\0\0GNU\0I\17\357\204\3$\f\221\2039x\324\224\323\236S"..., 68, 896) =
68

[pid  3416] newfstatat(4, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, AT_EMPTY_PATH) =
0

[pid  3416] pread64(4, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"...,
784, 64) = 784

[pid  3416] mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 4, 0) = 0x72f6fc000000

[pid  3416] mprotect(0x72f6fc028000, 2023424, PROT_NONE) = 0

[pid  3416] mmap(0x72f6fc028000, 1658880, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 4, 0x28000) = 0x72f6fc028000

[pid  3416] mmap(0x72f6fc1bd000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 4,
0x1bd000) = 0x72f6fc1bd000

[pid  3416] mmap(0x72f6fc216000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 4, 0x215000) = 0x72f6fc216000

[pid  3416] mmap(0x72f6fc21c000, 52816, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x72f6fc21c000

[pid  3416] close(4)                    = 0

[pid  3416] mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x72f6fc239000

[pid  3416] arch_prctl(ARCH_SET_FS, 0x72f6fc239740) = 0

[pid  3416] set_tid_address(0x72f6fc239a10) = 3416
```

```
[pid  3416] set_robust_list(0x72f6fc239a20, 24) = 0

[pid  3416] rseq(0x72f6fc23a0e0, 0x20, 0, 0x53053053) = 0

[pid  3416] mprotect(0x72f6fc216000, 16384, PROT_READ) = 0

[pid  3416] mprotect(0x620821b86000, 4096, PROT_READ) = 0

[pid  3416] mprotect(0x72f6fc285000, 8192, PROT_READ) = 0

[pid  3416] prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY})
= 0

[pid  3416] munmap(0x72f6fc23c000, 58047) = 0

[pid  3416] read(0, F  f

 <unfinished ...>

[pid  3415] <... read resumed>"F  f\n", 1024) = 5

[pid  3415] write(4, "F  f\n", 5)       = 5

[pid  3416] <... read resumed>"F  f\n", 1024) = 5

[pid  3415] read(7,  <unfinished ...>

[pid  3416] write(1, "f  f\n", 5)       = 5

[pid  3417] <... read resumed>"f  f\n", 1024) = 5

[pid  3416] read(0,  <unfinished ...>

[pid  3417] write(1, "f f\n", 4)        = 4

[pid  3415] <... read resumed>"f f\n", 1023) = 4

[pid  3417] read(0,  <unfinished ...>

[pid  3415] write(1, "Processed result: ", 18Processed result: ) = 18

[pid  3415] write(1, "f f", 3f f)          = 3

[pid  3415] write(1, "\n\n", 2


)         = 2

[pid  3415] write(1, "Enter your string or (Enter / CT"..., 51Enter your string or (Enter /
CTRL + D) for stop:

) = 51

[pid  3415] read(0,

"\n", 1024)         = 1

[pid  3415] close(4)                    = 0

[pid  3416] <... read resumed>"", 1024) = 0

[pid  3415] close(7)                    = 0

[pid  3416] exit_group(0 <unfinished ...>
```

```
[pid  3415] close(6)                    = 0

[pid  3415] wait4(-1,  <unfinished ...>

[pid  3416] <... exit_group resumed>)   = ?

[pid  3417] <... read resumed>"", 1024) = 0

[pid  3416] +++ exited with 0 +++

[pid  3415] <... wait4 resumed>NULL, 0, NULL) = 3416

[pid  3417] exit_group(0 <unfinished ...>

[pid  3415] --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=3416, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---

[pid  3415] wait4(-1,  <unfinished ...>

[pid  3417] <... exit_group resumed>)   = ?

[pid  3417] +++ exited with 0 +++

<... wait4 resumed>NULL, 0, NULL)       = 3417

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=3417, si_uid=1000, si_status=0,
si_utime=0, si_stime=0} ---

exit_group(0)                           = ?

+++ exited with 0 +++
```

Вывод

Во время выполнения лабораторной работы я разработал программу, которая использует несколько процессов для обработки строк, вводимых пользователем. Основная сложность возникла из-за не закрытых каналов (pipes), что приводило к зависанию процессов: дочерние процессы не завершались, поскольку продолжали ждать ввода. Я исправил это, убедившись, что все ненужные дескрипторы закрыты после их использования. В будущем хотелось бы уделить больше времени отладке и тестированию процессов, чтобы избежать подобных проблем. В целом, работа была полезной и помогла мне лучше понять взаимодействие между процессами.