

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-211Б-23

Студент: Лапин Д.А.

Преподаватель: Бахарев В.Д.

Оценка: \_\_\_\_\_

Дата: 30.11.24

Москва, 2024

# Постановка задачи

## Вариант 16.

Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программа (основной процесс) должен создать для решение задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или каналы (pipe). Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы. Родительский процесс создает дочерний процесс. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child проверяет строки на валидность правилу. Если строка соответствует правилу, то она выводится в стандартный поток вывода дочернего процесса, иначе в pipe2 выводится информация об ошибке. Родительский процесс полученные от child ошибки выводит в стандартный поток вывода. Правило проверки: строка должна оканчиваться на «.» или «;»

## Общий метод и алгоритм решения

Использованные системные вызовы:

1. `pid_t fork(void)`; – создает новый дочерний процесс, который является копией родительского. В родительском процессе возвращает PID дочернего процесса, а в дочернем возвращает 0.
2. `int pipe(int *fd)`; – создает однонаправленный канал для передачи данных между процессами. Массив `fd` содержит два дескриптора: `fd[0]` для чтения и `fd[1]` для записи.
3. `int open(const char *pathname, int flags, mode_t mode)`; – открывает файл по указанному пути с заданными флагами (`O_WRONLY`, `O_CREAT`, и т.д.) и правами доступа (`mode`). Возвращает файловый дескриптор.
4. `ssize_t read(int fd, void *buf, size_t count)`; – считывает до `count` байт данных из дескриптора `fd` в буфер `buf`. Возвращает количество считанных байт.
5. `ssize_t write(int fd, const void *buf, size_t count)`; – записывает до `count` байт из буфера `buf` в файловый дескриптор `fd`. Возвращает количество записанных байт.
6. `int close(int fd)`; – закрывает файловый дескриптор `fd`, освобождая связанные с ним ресурсы.
7. `int dup2(int oldfd, int newfd)`; – дублирует файловый дескриптор `oldfd`, перенаправляя его на `newfd`. Используется для перенаправления стандартных потоков.
8. `pid_t getpid(void)`; – возвращает идентификатор текущего процесса (PID).
9. `int execv(const char *path, char *const argv[])`; – заменяет текущий процесс новым, загружая исполняемый файл по указанному пути `path` с аргументами `argv`.

10. `ssize_t readlink(const char *path, char *buf, size_t bufsiz);` – считывает символическую ссылку `path` и записывает содержимое в `buf`. Используется для получения пути к текущему исполняемому файлу.

11. `int wait(int *status);` – ожидает завершения дочернего процесса и возвращает его статус через указатель `status`.

12. `int fcntl(int fd, int cmd, ...);` – управляет поведением файлового дескриптора. В данной работе используется для установки неблокирующего режима с флагом `O_NONBLOCK`.

13. `void exit(int status);` – завершает процесс с указанным кодом выхода `status`. Используется для обработки ошибок и завершения программы.

14. `int snprintf(char *str, size_t size, const char *format, ...);` – форматирует строку в буфер `str` размером `size`, возвращая длину записанных символов. Используется для создания сообщений перед выводом.

Программа делает так, чтобы родительский процесс и дочерний могли обмениваться данными через каналы (`pipe`), а дочерний процесс запускает клиентскую программу для записи данных в файл.

1. Сначала родитель спрашивает у пользователя имя файла, чтобы передать его дочернему процессу. Также он узнаёт путь к текущей программе с помощью `readlink`.

2. Затем родитель создаёт два канала: один для передачи строк в дочерний процесс (`channel_data`), другой для получения ошибок от дочернего процесса (`channel_errors`). После этого с помощью `fork` создаётся дочерний процесс:

- Родительский процесс читает строки от пользователя и отправляет их в `channel_data`, а ещё отслеживает ошибки из `channel_errors`.

- Дочерний процесс перенаправляет свои потоки `stdin` и `stderr` на каналы и запускает клиентскую программу через `execv`.

3. Клиентская программа проверяет строки, которые ей присылают: они должны заканчиваться на точку (.) или точку с запятой (;). Если всё нормально, строка записывается в файл. Если нет — ошибка отправляется обратно через `stderr`.

4. В конце родительский процесс ждёт, пока дочерний завершится, с помощью `wait`. Все открытые каналы закрываются перед завершением работы.

Таким образом, программа показывает, как процессы могут взаимодействовать между собой и как данные передаются через каналы.

## Код программы

### posix\_ipc-example-server.c

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

static char CLIENT_PROGRAM_NAME[] = "posix_ipc-example-client";

int main() {

    char file[4096];
    write(STDIN_FILENO, "Enter filename: ", 16);
    if (scanf("%s", file) <= 0){

        char mssg[1024];
        uint32_t len = snprintf(mssg, sizeof(mssg) - 1, "Enter filename
failed\n");
        write(STDERR_FILENO, mssg, len);
        exit(EXIT_SUCCESS);
    }

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

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

        prospath[len] = '\0';
    }
}
```

```

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

// NOTE: Spawn a new process
const pid_t child = fork();

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

    case 0: { // NOTE: We're a child, child doesn't know its pid after fork
        pid_t pid = getpid(); // NOTE: Get child PID
        close(channel_data[1]); // Unused in child
        close(channel_errors[0]); // Unused in child
        // NOTE: Connect parent stdin to child stdin
        dup2(channel_data[0], STDIN_FILENO); // Перенаправляем stdin на
        чтение из channel_data[0]
        dup2(channel_errors[1], STDERR_FILENO);

        {
            char msg[64];
            const int32_t length = snprintf(msg, sizeof(msg),
                                           "%d: I'm a child\n", pid);
            write(STDOUT_FILENO, msg, length);
        }

        {
            char path[1024];
            snprintf(path, sizeof(path) - 1, "%s/%s", proppath,
CLIENT_PROGRAM_NAME);
            // NOTE: args[0] must be a program name, next the actual arguments
            // NOTE: `NULL` at the end is mandatory, because `exec*`

```

```

// expects a NULL-terminated list of C-strings
char *const args[] = {CLIENT_PROGRAM_NAME, file, NULL};

int32_t status = execv(path, args);

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

} break;

default: { // NOTE: We're a parent, parent knows PID of child after fork
    pid_t pid = getpid(); // NOTE: Get parent PID
    char buf1[4096];
    char buf[4096];
    ssize_t bytes;
    ssize_t bytes1;

    close(channel_data[0]);
    close(channel_errors[1]);

    {
        char msg[64];
        const int32_t length = snprintf(msg, sizeof(msg),
                                         "%d: I'm a parent, my child has
PID %d\n", pid, child);
        write(STDOUT_FILENO, msg, length);
    }
    {
        char msg[128];
        int32_t len = snprintf(msg, sizeof(msg) - 1,
                                "%d: Start typing lines of text. Press
'Ctrl-D' or 'Enter' with no input to exit\n", pid);
        write(STDOUT_FILENO, msg, len);
    }

    fcntl(channel_errors[0], F_SETFL, O_NONBLOCK); // Устанавливаем
неблокирующий режим для channel_errors[0]

    while ((bytes1 = read(STDIN_FILENO, buf1, sizeof(buf1))) ) {
        // Читаем ошибки, если они есть
        ssize_t error_bytes = read(channel_errors[0], buf, sizeof(buf));
        if (error_bytes > 0) {
            char msg[64];
            int32_t length = snprintf(msg, sizeof(msg), "%d: I'm a parent,
my child has PID %d\n", pid, child);

```

```

        write(STDOUT_FILENO, msg, length);

        write(STDOUT_FILENO, buf, error_bytes); // Выводим только
реальное количество считанных байт
    }

    if (bytes1 < 0){
        const char msg[] = "error: failed to read from stdin\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }
    if (buf1[0] == '\n') {
        break; // Выход, если введена пустая строка
    }

    // Отправляем строку клиенту
    if (write(channel_data[1], buf1, bytes1) == -1) {
        const char msg[] = "error: failed to write to channel_data\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }

}

close(channel_data[1]);
close(channel_errors[0]);

// NOTE: `wait` blocks the parent until child exits
int child_status;
wait(&child_status);
if (EXIT_SUCCESS) {WIFEXITED(child_status) && WEXITSTATUS(child_status) !=
    const char msg[] = "error: child exited with error\n";
    write(STDERR_FILENO, msg, sizeof(msg) - 1);
    exit(WEXITSTATUS(child_status));
}

} break;
}
}

```

### posix\_ipc-example-client.c

```

#include <stdint.h>
#include <stdbool.h>
#include <string.h>

```

```

#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>

int main(int argc, char **argv) {
    char buf[4096];
    ssize_t bytes;

    // NOTE: `O_WRONLY` only enables file for writing
    // NOTE: `O_CREAT` creates the requested file if absent
    // NOTE: `O_TRUNC` empties the file prior to opening
    // NOTE: `O_APPEND` subsequent writes are being appended instead of
    // overwritten
    int32_t file = open(argv[1], O_WRONLY | O_CREAT | O_TRUNC | O_APPEND, 0600);
    if (file == -1) {
        const char msg[] = "error: failed to open requested file\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }

    while ((bytes = read(STDIN_FILENO, buf, sizeof(buf)))) {
        int8_t f = 1;
        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') {
            // NOTE: When Enter is pressed with no input, then exit client
            break;
        }

        {
            if (buf[bytes - 2] != ';' && buf[bytes - 2] != '.'){

                char msg[4096];
                int len = snprintf(msg, sizeof(msg), "child error: string does not
end with ; or . Error string: %s", buf);

                write(STDERR_FILENO, msg, len);
                f = 0;
            }
        }

        {

```



```

// NOTE: Replace newline with NULL-terminator
buf[bytes - 1] = '\n';
if (f != 0) {
    int32_t written = write(file, buf, bytes);
    if (written != bytes) {
        const char msg[] = "error: failed to write to file\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }
}
memset(buf, 0, sizeof(buf));
}

}

close(file);
}

```

## Протокол работы программы

### Тестирование:

qbzy@QBZstation:/mnt/c/Users/mrbor/CLionProjects/osi/lab1\$ ./posix\_ipc-example-server

Enter filename: f.txt

89801: I'm a parent, my child has PID 89802

89801: Start typing lines of text. Press 'Ctrl-D' or 'Enter' with no input to exit

89802: I'm a child

fkjsdlfjsdf;

hkjhdfsa.

fhsdaf

fdsa1

89801: I'm a parent, my child has PID 89802

child error: string does not end with ; or . Error string: fhsdaf

34234

89801: I'm a parent, my child has PID 89802

child error: string does not end with ; or . Error string: fdsa1

l.;

89801: I'm a parent, my child has PID 89802

child error: string does not end with ; or . Error string: 34234

l;l;

fjdlsaf.

fjdlskajf;

3429995j

jjl

89801: I'm a parent, my child has PID 89802

child error: string does not end with ; or . Error string: 3429995j

89801: I'm a parent, my child has PID 89802

child error: string does not end with ; or . Error string: jjl

f.txt:

fkjsdlfjsdf;

hkjhdfsa.

l.;

l;l;

fjdlsaf.

fjdlskajf;

## Strace:

```
qbzy@QBZstation:/mnt/c/Users/mrbor/CLionProjects/osi/lab1$ strace -f
./posix_ipc-example-server

execve("./posix_ipc-example-server", ["/posix_ipc-example-server"], 0x7ffd4b486228 /* 26
vars */) = 0

brk(NULL)                                = 0x55c67d115000

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ff9a8c80000

access("/etc/ld.so.preload", R_OK)        = -1 ENOENT (No such file or directory)

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

fstat(3, {st_mode=S_IFREG|0644, st_size=20115, ...}) = 0

mmap(NULL, 20115, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7ff9a8c7b000

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\0\3\0>\0\1\0\0\0\220\243\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

fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 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, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7ff9a8a69000

mmap(0x7ff9a8a91000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x28000) = 0x7ff9a8a91000

mmap(0x7ff9a8c19000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) =
0x7ff9a8c19000

mmap(0x7ff9a8c68000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1fe000) = 0x7ff9a8c68000

mmap(0x7ff9a8c6e000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1,
0) = 0x7ff9a8c6e000

close(3)                                  = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ff9a8a66000

arch_prctl(ARCH_SET_FS, 0x7ff9a8a66740) = 0

set_tid_address(0x7ff9a8a66a10)           = 316686

set_robust_list(0x7ff9a8a66a20, 24)       = 0

rseq(0x7ff9a8a67060, 0x20, 0, 0x53053053) = 0

mprotect(0x7ff9a8c68000, 16384, PROT_READ) = 0

mprotect(0x55c65a50f000, 4096, PROT_READ) = 0
```

```

    mprotect(0x7ff9a8cb8000, 8192, PROT_READ) = 0

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

    munmap(0x7ff9a8c7b000, 20115)                = 0

    write(0, "Enter filename: ", 16Enter filename: )    = 16

fstat(0, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x5), ...}) = 0

getrandom("\x71\xa5\x96\xed\xdf\x8c\x6b\x63", 8, GRND_NONBLOCK) = 8

    brk(NULL)                                    = 0x55c67d115000

    brk(0x55c67d136000)                         = 0x55c67d136000

        read(0, f.txt
            "f.txt\n", 1024)                    = 6

readlink("/proc/self/exe", "/mnt/c/Users/mrbor/CLionProjects"... , 1023) = 66

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

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

clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLDstrace: Process
    316745 attached

    , child_tidptr=0x7ff9a8a66a10) = 316745

[pid 316745] set_robust_list(0x7ff9a8a66a20, 24 <unfinished ...>

    [pid 316686] getpid( <unfinished ...>

[pid 316745] <... set_robust_list resumed>) = 0

[pid 316686] <... getpid resumed>          = 316686

    [pid 316745] getpid( <unfinished ...>

    [pid 316686] close(3 <unfinished ...>

[pid 316745] <... getpid resumed>          = 316745

    [pid 316686] <... close resumed>           = 0

    [pid 316745] close(4 <unfinished ...>

    [pid 316686] close(6 <unfinished ...>

[pid 316745] <... close resumed>           = 0

[pid 316686] <... close resumed>           = 0

    [pid 316745] close(5 <unfinished ...>

[pid 316686] write(1, "316686: I'm a parent, my child h"... , 46 <unfinished ...>

    316686: I'm a parent, my child has PID 316745

    [pid 316745] <... close resumed>           = 0

[pid 316686] <... write resumed>           = 46

```

```

        [pid 316745] dup2(3, 0 <unfinished ...>

[pid 316686] write(1, "316686: Start typing lines of te"..., 84 <unfinished ...>

        [pid 316745] <... dup2 resumed>)                = 0
316686: Start typing lines of text. Press 'Ctrl-D' or 'Enter' with no input to exit

        [pid 316686] <... write resumed>)                = 84
        [pid 316745] dup2(6, 2 <unfinished ...>

[pid 316686] fcntl(5, F_SETFL, O_RDONLY|O_NONBLOCK <unfinished ...>

        [pid 316745] <... dup2 resumed>)                = 2
        [pid 316686] <... fcntl resumed>)                = 0

[pid 316745] write(1, "316745: I'm a child\n", 20 <unfinished ...>

        316745: I'm a child

        [pid 316686] read(0, <unfinished ...>

        [pid 316745] <... write resumed>)                = 20

[pid 316745] execve("/mnt/c/Users/mrbor/CLionProjects/osi/lab1/posix_ipc-example-client",
        ["posix_ipc-example-client", "f.txt"], 0x7ffd0a6a81c8 /* 26 vars */) = 0

        [pid 316745] brk(NULL)                            = 0x555c161cd000

[pid 316745] mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
        0x7f6ec4692000

[pid 316745] access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)

        [pid 316745] openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 4

        [pid 316745] fstat(4, {st_mode=S_IFREG|0644, st_size=20115, ...}) = 0

[pid 316745] mmap(NULL, 20115, PROT_READ, MAP_PRIVATE, 4, 0) = 0x7f6ec468d000

        [pid 316745] close(4)                            = 0

[pid 316745] openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 4

        [pid 316745] read(4,
        "\177ELF\2\1\1\3\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\243\2\0\0\0\0"..., 832) = 832
[pid 316745] 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 316745] fstat(4, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0

[pid 316745] 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 316745] mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 4, 0) =
        0x7f6ec447b000

        [pid 316745] mmap(0x7f6ec44a3000, 1605632, PROT_READ|PROT_EXEC,
        MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 4, 0x28000) = 0x7f6ec44a3000

```

```

[pid 316745] mmap(0x7f6ec462b000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 4,
                  0x1b0000) = 0x7f6ec462b000

[pid 316745] mmap(0x7f6ec467a000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 4, 0x1fe000) = 0x7f6ec467a000

[pid 316745] mmap(0x7f6ec4680000, 52624, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f6ec4680000

[pid 316745] close(4) = 0

[pid 316745] mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
                  0x7f6ec4478000

[pid 316745] arch_prctl(ARCH_SET_FS, 0x7f6ec4478740) = 0

[pid 316745] set_tid_address(0x7f6ec4478a10) = 316745

[pid 316745] set_robust_list(0x7f6ec4478a20, 24) = 0

[pid 316745] rseq(0x7f6ec4479060, 0x20, 0, 0x53053053) = 0

[pid 316745] mprotect(0x7f6ec467a000, 16384, PROT_READ) = 0

[pid 316745] mprotect(0x555bd9c55000, 4096, PROT_READ) = 0

[pid 316745] mprotect(0x7f6ec46ca000, 8192, PROT_READ) = 0

[pid 316745] prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024,
                  rlim_max=RLIM64_INFINITY}) = 0

[pid 316745] munmap(0x7f6ec468d000, 20115) = 0

[pid 316745] openat(AT_FDCWD, "f.txt", O_WRONLY|O_CREAT|O_TRUNC|O_APPEND, 0600) = 4

[pid 316745] read(0, fdkfdjsa;

<unfinished ...>

[pid 316686] <... read resumed>"fdkfdjsa;\n", 4096) = 10

[pid 316686] read(5, 0x7ffd0a6a7080, 4096) = -1 EAGAIN (Resource temporarily unavailable)

[pid 316686] write(4, "fdkfdjsa;\n", 10) = 10

[pid 316745] <... read resumed>"fdkfdjsa;\n", 4096) = 10

[pid 316686] read(0, <unfinished ...>

[pid 316745] write(4, "fdkfdjsa;\n", 10) = 10

[pid 316745] read(0, jkjfdsfs.

<unfinished ...>

[pid 316686] <... read resumed>"jkjfdsfs.\n", 4096) = 10

[pid 316686] read(5, 0x7ffd0a6a7080, 4096) = -1 EAGAIN (Resource temporarily unavailable)

[pid 316686] write(4, "jkjfdsfs.\n", 10) = 10

[pid 316745] <... read resumed>"jkjfdsfs.\n", 4096) = 10

[pid 316686] read(0, <unfinished ...>

```

```

[pid 316745] write(4, "jkjfdsfs.\n", 10) = 10
[pid 316745] read(0, fjsdfsd1
<unfinished ...>
[pid 316686] <... read resumed>"fjsdfsd1\n", 4096) = 9
[pid 316686] read(5, 0x7ffd0a6a7080, 4096) = -1 EAGAIN (Resource temporarily unavailable)
[pid 316686] write(4, "fjsdfsd1\n", 9) = 9
[pid 316745] <... read resumed>"fjsdfsd1\n", 4096) = 9
[pid 316686] read(0, <unfinished ...>
[pid 316745] write(2, "child error: string does not end"..., 68) = 68
[pid 316745] read(0, 2343245
<unfinished ...>
[pid 316686] <... read resumed>"2343245\n", 4096) = 8
[pid 316686] read(5, "child error: string does not end"..., 4096) = 68
[pid 316686] write(1, "316686: I'm a parent, my child h"..., 46316686: I'm a parent, my
child has PID 316745
) = 46
[pid 316686] write(1, "child error: string does not end"..., 68child error: string does not
end with ; or . Error string: fjsdfsd1
) = 68
[pid 316686] write(4, "2343245\n", 8) = 8
[pid 316745] <... read resumed>"2343245\n", 4096) = 8
[pid 316686] read(0, <unfinished ...>
[pid 316745] write(2, "child error: string does not end"..., 67) = 67
[pid 316745] read(0, 23423442345
<unfinished ...>
[pid 316686] <... read resumed>"23423442345\n", 4096) = 12
[pid 316686] read(5, "child error: string does not end"..., 4096) = 67
[pid 316686] write(1, "316686: I'm a parent, my child h"..., 46316686: I'm a parent, my
child has PID 316745
) = 46
[pid 316686] write(1, "child error: string does not end"..., 67child error: string does not
end with ; or . Error string: 2343245
) = 67
[pid 316686] write(4, "23423442345\n", 12) = 12
[pid 316745] <... read resumed>"23423442345\n", 4096) = 12

```

```

        [pid 316686] read(0, <unfinished ...>

[pid 316745] write(2, "child error: string does not end"..., 71) = 71

        [pid 316745] read(0, kfjs;da;

                <unfinished ...>

[pid 316686] <... read resumed>"kfjs;da;\n", 4096) = 9

[pid 316686] read(5, "child error: string does not end"..., 4096) = 71

[pid 316686] write(1, "316686: I'm a parent, my child h"..., 46316686: I'm a parent, my
        child has PID 316745

                ) = 46

[pid 316686] write(1, "child error: string does not end"..., 71child error: string does not
        end with ; or . Error string: 23423442345

                ) = 71

[pid 316686] write(4, "kfjs;da;\n", 9) = 9

[pid 316745] <... read resumed>"kfjs;da;\n", 4096) = 9

        [pid 316686] read(0, <unfinished ...>

[pid 316745] write(4, "kfjs;da;\n", 9) = 9

        [pid 316745] read(0, fjs,.

                <unfinished ...>

[pid 316686] <... read resumed>"fjs,.\n", 4096) = 6

[pid 316686] read(5, 0x7ffd0a6a7080, 4096) = -1 EAGAIN (Resource temporarily unavailable)

        [pid 316686] write(4, "fjs,.\n", 6) = 6

[pid 316745] <... read resumed>"fjs,.\n", 4096) = 6

        [pid 316686] read(0, <unfinished ...>

[pid 316745] write(4, "fjs,.\n", 6) = 6

        [pid 316745] read(0,

                <unfinished ...>

[pid 316686] <... read resumed>"\\n", 4096) = 1

[pid 316686] read(5, 0x7ffd0a6a7080, 4096) = -1 EAGAIN (Resource temporarily unavailable)

        [pid 316686] close(4) = 0

[pid 316745] <... read resumed>"", 4096) = 0

        [pid 316686] close(5 <unfinished ...>

        [pid 316745] close(4 <unfinished ...>

[pid 316686] <... close resumed>) = 0

[pid 316686] wait4(-1, <unfinished ...>

```



```

[pid 316745] <... close resumed>          = 0

[pid 316745] exit_group(0)                = ?

[pid 316745] +++ exited with 0 +++

<... wait4 resumed>[{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 316745
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=316745, si_uid=1000, si_status=0,
      si_utime=0, si_stime=0} ---

lseek(0, -1, SEEK_CUR)                    = -1 ESPIPE (Illegal seek)

exit_group(0)                             = ?

+++ exited with 0 +++

```

## Вывод

**Программа успешно реализована и протестирована. Все требования задания выполнены:**

- Родительский и дочерний процессы реализованы в виде отдельных программ.
- Взаимодействие между процессами осуществлено через каналы pipe.
- Проверка строк на валидность выполнена в дочернем процессе.
- Системные ошибки обработаны.
- Результаты обработки корректно выводятся в стандартные потоки вывода родительского и дочернего процессов.