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-210БВ-24

Студент: Кудряшова Т.И.

Преподаватель: Бахарев В.Д.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: 02.10.25

Москва, 2025

**Постановка задачи**

**Вариант 10.**

Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия файла с таким именем на чтение. Стандартный поток ввода дочернего процесса переопределяется открытым файлом. Дочерний процесс читает команды из стандартного потока ввода. Стандартный поток вывода дочернего процесса перенаправляется в pipe1. Родительский процесс читает из pipe1 и прочитанное выводит в свой стандартный поток вывода. Родительский и дочерний процесс должны быть представлены разными программами.

В файле записаны команды вида: «число». Дочерний процесс производит проверку этого числа на простоту. Если число составное, то дочерний процесс пишет это число в стандартный поток вывода. Если число отрицательное или простое, то тогда дочерний и родительский процессы завершаются. Количество чисел может быть произвольным.

**Общий метод и алгоритм решения**

**Кратко опишите системные вызовы, которые вы использовали в лабораторной работе.**

Использованные системные вызовы:

* pid\_t fork(void); – создает дочерний процесс.
* readlink("/proc/self/exe", progpath, sizeof(progpath) - 1) – получение пути к текущей дирректории.
* int pipe(int \*fd); – открытие pipe.
* close(client\_to\_server[1]); - закрытие pipe.
* write(STD…\_FILENO, msg, sizeof(msg)) – запись в поток STD…\_FILENO.
* bytes = read(STD…\_FILENO, buf, sizeof(buf) - 1); - чтение из потока STD…\_FILENO.
* STDERR\_FILENO – поток для ошибок.
* STDIN\_FILENO – поток IN.
* STDOUT\_FILENO - поток OUT.

В рамках решения задачи алгоритм программы выглядит так:

1. Получение пути к текущей директории.
2. Открытие каналов между родителем и ребенком, а также ребенком и родителем.
3. Ответвление ребенка.
4. Чтение названия файла из stdin.
5. Открытие файла и заброс его содержимого на поток ввода процесса child.

Далее действия в child.

1. Расшифровка полученных данных.
2. Вывод составных чисел на поток вывода.
3. При получении простого или отрицательного - окончание программы.

**Код программы**

**parent.c**

#include <stdint.h>

#include <stdbool.h>

#include <unistd.h>

#include <sys/wait.h>

#include <stdlib.h>

#include <stdio.h>

#include <ctype.h>

#include <string.h>

#include <fcntl.h>

static char SERVER\_PROGRAM\_NAME[] = "child";

int main(int argc, char \*\*argv) {

    // NOTE: Get full path to the directory, where program resides

    char progpath[1024];

    {

        // NOTE: 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);

        }

        // NOTE: Trim the path to first slash from the end

        while (progpath[len] != '/')

            --len;

        progpath[len] = '\0';

    }

    // NOTE: Open pipes

    int client\_to\_server[2]; // AB

    if (pipe(client\_to\_server) == -1) {

        const char msg[] = "error: failed to create pipe\n";

        write(STDERR\_FILENO, msg, sizeof(msg));

        exit(EXIT\_FAILURE);

    }

    int server\_to\_client[2]; // BA

    if (pipe(server\_to\_client) == -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

        close(client\_to\_server[1]);

        close(server\_to\_client[0]);

        dup2(client\_to\_server[0], STDIN\_FILENO);

        close(client\_to\_server[0]);

        dup2(server\_to\_client[1], STDOUT\_FILENO);

        close(server\_to\_client[1]);

        {

            char path[1024];

            snprintf(path, sizeof(path) - 1, "%s/%s", progpath, SERVER\_PROGRAM\_NAME);

            // NOTE: Start server without extra args; will read filename from stdin

            char \*const args[] = {SERVER\_PROGRAM\_NAME, NULL};

            int32\_t status = execv(path, 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;

    default: { // NOTE: We're a parent, parent knows PID of child after fork

        close(client\_to\_server[0]);

        close(server\_to\_client[1]);

        char buf[4096];

        ssize\_t bytes;

        // 1) Read filename from our stdin

        bytes = read(STDIN\_FILENO, buf, sizeof(buf) - 1);

        if (bytes <= 0) {

            const char msg[] = "error: failed to read filename from stdin\n";

            write(STDERR\_FILENO, msg, sizeof(msg) - 1);

            exit(EXIT\_FAILURE);

        }

        buf[bytes] = '\0';

        char \*nl = strchr(buf, '\n');

        if (nl) \*nl = '\0';

        // 2) Open file and stream its contents into server stdin

        int fd = open(buf, O\_RDONLY);

        if (fd == -1) {

            const char msg[] = "error: failed to open input file\n";

            write(STDERR\_FILENO, msg, sizeof(msg) - 1);

            exit(EXIT\_FAILURE);

        }

        while ((bytes = read(fd, buf, sizeof(buf))) > 0) {

            if (write(client\_to\_server[1], buf, bytes) != bytes) {

                const char msg[] = "error: failed to write to server stdin\n";

                write(STDERR\_FILENO, msg, sizeof(msg) - 1);

                close(fd);

                exit(EXIT\_FAILURE);

            }

        }

        close(fd);

        close(client\_to\_server[1]); // signal EOF to server

        // 3) Read server response until EOF and print to stdout

        while ((bytes = read(server\_to\_client[0], buf, sizeof(buf))) > 0) {

            write(STDOUT\_FILENO, buf, bytes);

        }

        close(server\_to\_client[0]);

        wait(NULL);

    } break;

    }

}

**Child.c**

#include <stdint.h>

#include <stdbool.h>

#include <ctype.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <stdio.h>

#include <string.h>

int is\_composit(long long x) {

    if (x < 2) return 0;

    for (long long i = 2; i \* i <= x; ++i) {

        if (x % i == 0) return 1;

    }

    return 0;

}

int main(int argc, char \*\*argv) {

    char buf[4096];

    ssize\_t bytes;

    // Read numbers line by line from stdin

    char line[256];

    uint32\_t linelen = 0;

    while ((bytes = read(STDIN\_FILENO, buf, sizeof(buf))) > 0) {

        for (uint32\_t i = 0; i < (uint32\_t)bytes; ++i) {

            char c = buf[i];

            if (c == '\n') {

                line[linelen] = '\0';

                // Parse integer

                char \*endptr = NULL;

                long long value = strtoll(line, &endptr, 10);

                if (endptr != line) {

                    if (!is\_composit(value)) {

                        exit(EXIT\_SUCCESS);

                    }

                    char out[64];

                    int32\_t outlen = snprintf(out, sizeof(out), "%lld\n", value);

                    if (write(STDOUT\_FILENO, out, outlen) != outlen) {

                        const char msg[] = "error: failed to echo\n";

                        write(STDERR\_FILENO, msg, sizeof(msg) - 1);

                        exit(EXIT\_FAILURE);

                    }

                }

                linelen = 0;

            } else {

                if (linelen + 1 < sizeof(line)) {

                    line[linelen++] = c;

                } else {

                    // Line too long, reset to avoid overflow

                    linelen = 0;

                }

            }

        }

    }

    // If the last line doesn't end with newline, process it as well

    if (linelen > 0) {

        line[linelen] = '\0';

        char \*endptr = NULL;

        long long value = strtoll(line, &endptr, 10);

        if (endptr != line) {

            if (!is\_composit(value)) {

                exit(EXIT\_SUCCESS);

            }

            char out[64];

            int32\_t outlen = snprintf(out, sizeof(out), "%lld\n", value);

            write(STDOUT\_FILENO, out, outlen);

        }

    }

    if (bytes < 0) {

        const char msg[] = "error: failed to read from stdin\n";

        write(STDERR\_FILENO, msg, sizeof(msg) - 1);

        exit(EXIT\_FAILURE);

    }

}

**Протокол работы программы**

Тест 1:

Содержание 1.txt:

10

8

5

10

В консоли:

./parent filename

1.txt

Вывод:

10

8

Тест 2:

Содержание 2.txt:

4

25

-5

В консоли:

./parent filename

2.txt

Вывод:

4

25

**Вывод**

Было сложно разобраться в задании, а точнее теоретической подоплеке этих процессов.