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-216Б-23

Студент: Ермакова М. П.

Преподаватель: Бахарев В.Д.

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

Дата: 10.10.24

Москва, 2024

**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.

10 вариант) В файле записаны команды вида: «число<endline>». 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**

Использованные системные вызовы:

* pid\_t fork(void); – создает дочерний процесс.
* int pipe(int \*fd); – создание неименованного канала для передачи данных между

процессами

* void exit(int status) - завершения выполнения процесса и возвращение статуса
* int execl(const char \*filename, char \*const argv[], char \*const envp[]) - замена образа памяти процесса
* int dup2(int oldfd, int newfd) - переназначение файлового дескриптора
* int open(const char \*pathname, int flags, mode\_t mode) - открытие\создание файла
* int close(int fd) - закрыть файл
* pid\_t wait(int \*status) - Ожидание завершения дочернего процесса

В рамках лабораторной работы была разработана многопроцессная система для обработки числовых данных из файла. Программа состоит из двух частей: родительского процесса, который управляет вводом-выводом и координацией, и дочернего процесса, выполняющего фильтрацию чисел.

Родительский процесс начинает работу с запроса имени файла у пользователя. После получения имени он открывает указанный файл для чтения и создает неименованный канал (pipe) для межпроцессного взаимодействия. Затем с помощью системного вызова fork() создается дочерний процесс. В дочернем процессе происходит перенаправление стандартных потоков ввода-вывода: стандартный ввод связывается с открытым файлом, а стандартный вывод - с записывающей стороной канала. После этого дочерний процесс заменяет свой образ на программу-обработчик с помощью execl().

Дочерний процесс представляет собой программу-фильтр, которая читает числа из стандартного ввода и проверяет их на простоту. Если число оказывается простым или отрицательным, дочерний процесс завершает работу. В противном случае число выводится в стандартный вывод, который перенаправлен в канал.

Родительский процесс тем временем читает данные из читающей стороны канала и выводит их пользователю. После завершения передачи данных родительский процесс ожидает завершения дочернего процесса с помощью wait() и корректно закрывает все файловые дескрипторы.

**Код программы**

**parent.c**

#include <unistd.h>

#include <stdlib.h>

#include <fcntl.h>

#include <sys/wait.h>

int main() {

    char file\_name[200];

    const char message[] = "Enter name of file: ";

    write(STDOUT\_FILENO, message, sizeof(message));

    size\_t len\_name  = read(STDIN\_FILENO, file\_name, sizeof(file\_name) - 1);

    if (len\_name <= 0) {

        const char mess[] = "Error reading the file\n";

        write(STDOUT\_FILENO, mess, sizeof(mess));

        exit(EXIT\_FAILURE);

    }

    if (file\_name[len\_name - 1] == '\n') {

            file\_name[len\_name - 1] = '\0';

    } else file\_name[len\_name] = '\0';

    int file = open(file\_name, O\_RDONLY);

    if (file == -1) {

        const char mess[] = "Error opening file\n";

        write(STDOUT\_FILENO, mess, sizeof(mess));

        exit(EXIT\_FAILURE);

    }

    int child\_to\_parent[2];

    if (pipe(child\_to\_parent) == -1) {

        const char mess[] = "Error create pipe\n";

        write(STDOUT\_FILENO, mess, sizeof(mess));

        exit(EXIT\_FAILURE);

    }

    const pid\_t child = fork();

    switch (child)

    {

    case -1:

    {   const char msg[] = "error: failed to spawn new process\n";

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

        exit(EXIT\_FAILURE);

    }

        break;

    case 0:

    {

        close(child\_to\_parent[0]);

        dup2(child\_to\_parent[1], STDOUT\_FILENO);

        close(child\_to\_parent[1]);

        dup2(file, STDIN\_FILENO);

        close(file);

        execl("./child", "child", NULL);

        const char mess[] = "Error executing child\n";

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

        exit(EXIT\_FAILURE);

    }

    default:

    {

        close(file);

        close(child\_to\_parent[1]);

        char buf[100];

        size\_t len;

        while((len = read(child\_to\_parent[0], buf, sizeof(buf))) > 0) {

            write(STDOUT\_FILENO, buf, len);

        }

        close(child\_to\_parent[0]);

        wait(NULL);

    }

        break;

    }

    return 0;

}

**child.c**

#include <stdbool.h>

#include <unistd.h>

#include <stdlib.h>

#include <stdio.h>

bool is\_prime(int num) {

    if (num < 2) return false;

    if (num == 2) return true;

    for (int i = 2; i \* i <= num; i++) {

        if (num % i == 0) {

            return false;

        }

    }

    return true;

}

int read\_num() {

    int number = 0;

    bool start = false;

    char c;

    size\_t len;

    char flag = 1;

    while((len = read(STDIN\_FILENO, &c, 1)) > 0) {

        if (c == '-') {

            flag = -1;

        } else if (c >= '0' && c <= '9') {

            number = number \* 10 + (c - '0');

            start = true;

        } else if (c == '\n' || c == ' ' || c == '\t' || c == '\r') {

            if (start) {

                return number \* flag;

            }

        } else {

            printf("s = %c", c);

            const char mess[] = "Error: invalid character in input\n";

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

            \_exit(EXIT\_FAILURE);

        }

    }

    if (len == 0 && start) {

        return number \* flag;

    }

    return -1;

}

void write\_num(int n) {

    char buffer[32];

    int len = snprintf(buffer, sizeof(buffer), "%d\n", n);

    write(STDOUT\_FILENO, buffer, len);

    //write(STDOUT\_FILENO, "\n", 1);

}

int main() {

    int number;

    while(1) {

        number = read\_num();

        if (is\_prime(number) || number < 0) {

            \_exit(0);

        } else write\_num(number);

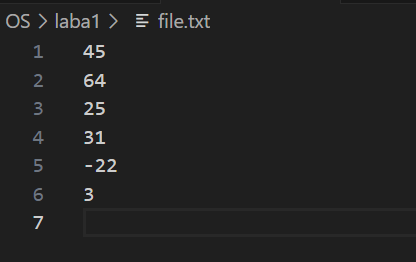
    }

    return 0;

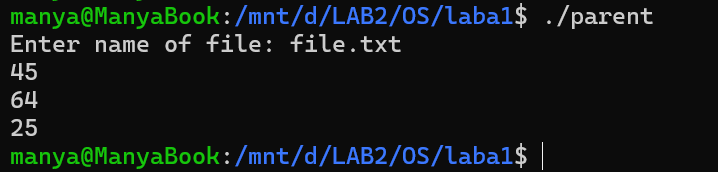
}

**Протокол работы программы**

**Содержимое файла file.txt:**

****

**Вывод программы:**

****

**Тестирование:**

$ ./parent

File.txt

**Strace:**

$ strace -f -o trace.txt ./parent

pipe2([4, 5], 0) = 0

clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7f4e36201a10) = 5352

**Вывод**

Программа корректно создает дочерний процесс и организует передачу данных между процессами с помощью pipe, что подтверждается анализом системных вызовов strace.

Основные проблемы при выполнении работы возникли с пониманием принципов работы с каналами pipe, особенно в области правильного закрытия файловых дескрипторов и организации двунаправленной коммуникации между процессами. Также сложности вызвало перенаправление стандартных потоков ввода-вывода и синхронизация работы родительского и дочернего процессов.