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-209Б-23

Студент: Корепанов И.А.

Преподаватель: Миронов Е.С.

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

Дата: 06.10.25

Москва, 2025

**Постановка задачи**

**Вариант 10.**

**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**

Использованные системные вызовы:

pid\_t fork(void); – создает дочерний процесс.

int pipe(int \*fd); – создание неименованного канала для передачи данных между процессами

int dup2(int oldfd, int newfd); — переназначение файлового дескриптора

int execl(const char \*path, const char \*arg, …, (char \*)0); — замещает текущий процесс новой программой.

int close(int fd); — закрывает файловый дескриптор, освобождая системные ресурсы.

Сначала были созданы необходимые переменные, включая файловые дескрипторы, которые использовались для организации обмена данными между процессами. После этого родительский процесс запрашивал у пользователя имя файла и открывал его для чтения. Затем создавался неименованный канал (pipe), обеспечивающий связь между родительским и дочерним процессами. После вызова fork() создавался дочерний процесс, в котором стандартный поток ввода перенаправлялся на открытый файл, а стандартный поток вывода — в конец записи канала. Далее с помощью системного вызова execl() запускалась отдельная программа дочернего процесса. Дочерний процесс считывал числа из файла, проверял их на простоту и передавал составные числа в канал. Родительский процесс, в свою очередь, читал данные из канала и выводил их на экран. После завершения работы дочернего процесса родитель закрывал все ненужные файловые дескрипторы и завершал выполнение программы.

**Код программы**

**parent.c**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

#define MAXSIZE 256

int main() {

    int pipefd[2];

    pid\_t pid;

    char filename[MAXSIZE];

    if (pipe(pipefd) == -1) {

        perror("pipe");

        exit(EXIT\_FAILURE);

    }

    printf("файл:");

    if (scanf("%255s", filename) != 1) {

        fprintf(stderr, "неверное имя файла\n");

        exit(EXIT\_FAILURE);

    }

    pid = fork();

    if (pid == -1) {

        perror("fork");

        exit(EXIT\_FAILURE);

    }

    if (pid == 0) {

        close(pipefd[0]);

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

        close(pipefd[1]);

        execl("./child", "child", filename, (char \*)NULL);

        perror("execl");

        exit(EXIT\_FAILURE);

    } else {

        close(pipefd[1]);

        char \*buffer = (char \*)malloc(256);

        int buffer\_size = 256;

        int total\_read = 0;

        ssize\_t count;

        while ((count = read(pipefd[0], buffer + total\_read, buffer\_size - total\_read - 1)) > 0) {

            total\_read += count;

            if (total\_read >= buffer\_size - 1) {

                buffer\_size \*= 2;

                char \*temp = (char \*)realloc(buffer, buffer\_size);

                if (temp == NULL) {

                    free(buffer);

                    perror("realloc");

                    exit(EXIT\_FAILURE);

                }

                buffer = temp;

            }

        }

        if (count == -1) {

            perror("read");

            free(buffer);

            exit(EXIT\_FAILURE);

        }

        buffer[total\_read] = '\0';

        printf("%s", buffer);

        fflush(stdout);

        if (atoi(buffer) <= 0) {

        }

        free(buffer);

        close(pipefd[0]);

        wait(NULL);

    }

    return 0;

}

**child.c**#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <string.h>

int isPrime(int n) {

    if (n <= 1) return 0;

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

        if (n % i == 0) return 0;

    }

    return 1;

}

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

    if (argc != 2) {

        fprintf(stderr, argv[0]);

        exit(EXIT\_FAILURE);

    }

    int fd = open(argv[1], O\_RDONLY);

    if (fd == -1) {

        perror("open");

        exit(EXIT\_FAILURE);

    }

    dup2(fd, STDIN\_FILENO);

    close(fd);

    char \*buffer = (char \*)malloc(256);

    int buffer\_size = 256;

    int total\_read = 0;

    while (1) {

        if (fgets(buffer + total\_read, buffer\_size - total\_read, stdin) != NULL) {

            total\_read += strlen(buffer + total\_read);

            if (total\_read >= buffer\_size - 1 || buffer[total\_read - 1] == '\n') {

                int num = atoi(buffer);

                if (num <= 0) {

                    free(buffer);

                    exit(EXIT\_SUCCESS);

                }

                if (!isPrime(num)) {

                    printf("%d\n", num);

                    fflush(stdout);

                } else {

                    free(buffer);

                    exit(EXIT\_SUCCESS);

                }

                total\_read = 0;

            } else if (total\_read >= buffer\_size - 1) {

                buffer\_size \*= 2;

                char \*temp = (char \*)realloc(buffer, buffer\_size);

                if (temp == NULL) {

                    free(buffer);

                    perror("realloc");

                    exit(EXIT\_FAILURE);

                }

                buffer = temp;

            }

        } else {

            break;

        }

    }

    free(buffer);

    return 0;

}

**Протокол работы программы**

./parent

файл:input.txt

44

6

$ strace ./parent

strace ./parent

execve("./parent", ["./parent"], 0x7ffebbbfe220 /\* 29 vars \*/) = 0

brk(NULL) = 0x6347694a8000

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=21784, ...}) = 0

mmap(NULL, 21784, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7a152897f000

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\0\260>\2\0\0\0\0\0"..., 832) = 832

fstat(3, {st\_mode=S\_IFREG|0755, st\_size=1901536, ...}) = 0

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7a152897d000

mmap(NULL, 1914496, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7a15287a9000

mmap(0x7a15287cb000, 1413120, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x22000) = 0x7a15287cb000

mmap(0x7a1528924000, 323584, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x17b000) = 0x7a1528924000

mmap(0x7a1528973000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1c9000) = 0x7a1528973000

mmap(0x7a1528979000, 13952, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7a1528979000

close(3) = 0

arch\_prctl(ARCH\_SET\_FS, 0x7a152897e540) = 0

mprotect(0x7a1528973000, 16384, PROT\_READ) = 0

mprotect(0x63474562c000, 4096, PROT\_READ) = 0

mprotect(0x7a15289af000, 4096, PROT\_READ) = 0

munmap(0x7a152897f000, 21784) = 0

**pipe([3, 4]) = 0**

fstat(1, {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0), ...}) = 0

brk(NULL) = 0x6347694a8000

brk(0x6347694c9000) = 0x6347694c9000

fstat(0, {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0), ...}) = 0

write(1, "\321\204\320\260\320\271\320\273:", 9файл:) = 9

read(0, input.txt

"input.txt\n", 1024) = 10

**clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7a152897e810) = 1616**

close(4) = 0

read(3, "44\n6\n", 255) = 5

read(3, "", 250) = 0

write(1, "44\n6\n", 544

6

) = 5

close(3) = 0

wait4(-1, NULL, 0, NULL) = 1616

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=1616, 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**

В ходе выполнения лабораторной работы были получены практические навыки создания каналов (pipe) и работы с процессами. Стало понятнее, как происходит взаимодействие между родительским и дочерним процессами, а также как управлять их поведением через системные вызовы. В процессе выполнения возникла проблема с тем, что дочерние процессы не завершались, которая решилась закрытием дескрипторов на чтение.