

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<newline>». 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();
    }
}

```

```

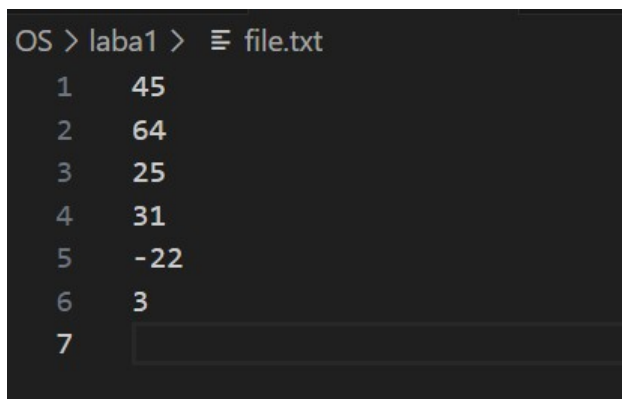
        if (is_prime(number) || number < 0) {
            _exit(0);
        } else write_num(number);
    }

    return 0;
}

```

## Протокол работы программы

Содержимое файла file.txt:

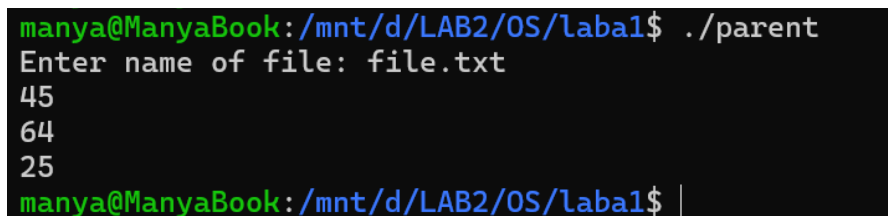


```

OS > laba1 > ≡ file.txt
1    45
2    64
3    25
4    31
5   -22
6     3
7

```

Вывод программы:



```

manya@ManyaBook:/mnt/d/LAB2/OS/laba1$ ./parent
Enter name of file: file.txt
45
64
25
manya@ManyaBook:/mnt/d/LAB2/OS/laba1$ |

```

Тестирование:

```

$ ./parent
File.txt

```

Strace:

```

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

```

```

5351  execve("./parent", [ "./parent" ], 0x7ffd8635fb78 /* 27 vars */) = 0
5351  brk(NULL)                                = 0x55cf05906000
5351  mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f4e3641b000
5351  access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
5351  openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

```

```

5351 fstat(3, {st_mode=S_IFREG|0644, st_size=19527, ...}) = 0
5351 mmap(NULL, 19527, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f4e36416000
5351 close(3) = 0
5351 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
832) 5351 read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\243\2\0\0\0\0\0"...
784, 5351) = 784 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"...
5351 fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0
784, 5351) = 784 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"...
5351 mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f4e36204000
5351 mmap(0x7f4e3622c000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x28000) = 0x7f4e3622c000
5351 mmap(0x7f4e363b4000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1b0000) = 0x7f4e363b4000
5351 mmap(0x7f4e36403000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x1fe000) = 0x7f4e36403000
5351 mmap(0x7f4e36409000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_ANONYMOUS, -1, 0) = 0x7f4e36409000
5351 close(3) = 0
5351 mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f4e36201000
5351 arch_prctl(ARCH_SET_FS, 0x7f4e36201740) = 0
5351 set_tid_address(0x7f4e36201a10) = 5351
5351 set_robust_list(0x7f4e36201a20, 24) = 0
5351 rseq(0x7f4e36202060, 0x20, 0, 0x53053053) = 0
5351 mprotect(0x7f4e36403000, 16384, PROT_READ) = 0
5351 mprotect(0x55cf019d5000, 4096, PROT_READ) = 0
5351 mprotect(0x7f4e36453000, 8192, PROT_READ) = 0
= 0 5351 prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY})
5351 munmap(0x7f4e36416000, 19527) = 0
5351 write(1, "Enter name of file: \0", 21) = 21
5351 read(0, "file.txt\n", 199) = 9
5351 openat(AT_FDCWD, "file.txt", O_RDONLY) = 3
5351 pipe2([4, 5], 0) = 0
5351 clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7f4e36201a10) = 5352
5352 set_robust_list(0x7f4e36201a20, 24 <unfinished ...>
5351 close(3 <unfinished ...>
5352 <... set_robust_list resumed>) = 0
5351 <... close resumed>) = 0
5352 close(4 <unfinished ...>
5351 close(5 <unfinished ...>
5352 <... close resumed>) = 0
5351 <... close resumed>) = 0
5352 dup2(5, 1 <unfinished ...>
5351 read(4, <unfinished ...>
5352 <... dup2 resumed>) = 1
5352 close(5) = 0
5352 dup2(3, 0) = 0
5352 close(3) = 0
5352 execve("./child", ["child"], 0x7fff6eae28 /* 27 vars */) = 0
5352 brk(NULL) = 0x55bf4518c000

```



```

5352 mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fa683631000
5352 access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
5352 openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
5352 fstat(3, {st_mode=S_IFREG|0644, st_size=19527, ...}) = 0
5352 mmap(NULL, 19527, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fa68362c000
5352 close(3) = 0
5352 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
832) 5352 read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\243\2\0\0\0\0\0"...
784, 64) = 784
5352 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
5352 fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0
784, 64) = 784
5352 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
5352 mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fa68341a000
5352 mmap(0x7fa683442000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x28000) = 0x7fa683442000
5352 mmap(0x7fa6835ca000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1b0000) = 0x7fa6835ca000
5352 mmap(0x7fa683619000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x1fe000) = 0x7fa683619000
5352 mmap(0x7fa68361f000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_ANONYMOUS, -1, 0) = 0x7fa68361f000
5352 close(3) = 0
0x7fa683417000 5352 mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
5352 arch_prctl(ARCH_SET_FS, 0x7fa683417740) = 0
5352 set_tid_address(0x7fa683417a10) = 5352
5352 set_robust_list(0x7fa683417a20, 24) = 0
5352 rseq(0x7fa683418060, 0x20, 0, 0x53053053) = 0
5352 mprotect(0x7fa683619000, 16384, PROT_READ) = 0
5352 mprotect(0x55bf15d6c000, 4096, PROT_READ) = 0
5352 mprotect(0x7fa683669000, 8192, PROT_READ) = 0
= 0 5352 prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY})
5352 munmap(0x7fa68362c000, 19527) = 0
5352 read(0, "4", 1) = 1
5352 read(0, "5", 1) = 1
5352 read(0, "\n", 1) = 1
5352 write(1, "45\n", 3 <unfinished ...>
5351 <... read resumed>"45\n", 100) = 3
5352 <... write resumed> = 3
5351 write(1, "45\n", 3 <unfinished ...>
5352 read(0, <unfinished ...>
5351 <... write resumed> = 3
5351 read(4, <unfinished ...>
5352 <... read resumed>"6", 1) = 1
5352 read(0, "4", 1) = 1
5352 read(0, "\n", 1) = 1
5352 write(1, "64\n", 3 <unfinished ...>
5351 <... read resumed>"64\n", 100) = 3
5352 <... write resumed> = 3
5351 write(1, "64\n", 3 <unfinished ...>
5352 read(0, <unfinished ...>

```

```

5351 <... write resumed>                = 3
5352 <... read resumed>"2", 1)          = 1
5351 read(4, <unfinished ...>
5352 read(0, "5", 1)                    = 1
5352 read(0, "\n", 1)                   = 1
5352 write(1, "25\n", 3 <unfinished ...>
5351 <... read resumed>"25\n", 100)     = 3
5352 <... write resumed>                = 3
5351 write(1, "25\n", 3 <unfinished ...>
5352 read(0, <unfinished ...>
5351 <... write resumed>                = 3
5352 <... read resumed>"3", 1)          = 1
5351 read(4, <unfinished ...>
5352 read(0, "1", 1)                    = 1
5352 read(0, "\n", 1)                   = 1
5352 exit_group(0)                      = ?
5351 <... read resumed>"", 100)         = 0
5351 close(4)                           = 0
5352 +++ exited with 0 +++
5351 --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=5352, si_uid=1000,
si_status=0, si_utime=0, si_stime=1 /* 0:01 s */} ---
5351 wait4(-1, NULL, 0, NULL)            = 5352
5351 exit_group(0)                      = ?
5351 +++ exited with 0 +++

```

## 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.