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-212БВ-24

Студент: Сучков В.С.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 14.11.25

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<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); – создает канал для однонаправленной передачи данных между процессами.
- close(int *fd); - используется для закрытия файлового дескриптора.
- dup2(int *fd1, int *fd2); - создает второй дескриптор файла для открытого файла.
- exit(int status); - завершение выполнения процесса и возвращение статуса.
- int execl(const char *filename, char *const argv[], char *const envp[]) - замена образа памяти процесса

Алгоритм решения:

A. Parent

1. **Запрос имени файла**
2. **Создание pipe**
3. **Создание дочернего процесса**
4. **Закрываем ненужные концы pipe**
5. **Чтение данных от дочернего процесса**
6. **Вывод данных пользователю**
7. **Обнаружение конца данных**
8. **Закрытие pipe**
9. **Ожидание завершения дочернего процесса**
10. **Завершение родительского процесса**

B. Child

1. **Получаем управление** после fork() с pid = 0
2. **Закрываем ненужные концы pipe**
3. **Перенаправляем стандартные потоки**
4. **Открываем файл для чтения**
5. **Выполняем замену программы**
6. **Инициализация**

7. **Чтение и обработка чисел**
8. **Обработка числа**
9. **Завершение**

Код программы

parent.c

```
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <fcntl.h>

int main() {
    char filename[256];
    ssize_t bytes_read;

    const char prompt[] = "Enter filename: ";
    write(STDOUT_FILENO, prompt, sizeof(prompt) - 1);

    bytes_read = read(STDIN_FILENO, filename, sizeof(filename) - 1);
    if (bytes_read <= 0) {
        const char error_msg[] = "Error reading filename\n";
        write(STDERR_FILENO, error_msg, sizeof(error_msg) - 1);
        exit(EXIT_FAILURE);
    }

    if (bytes_read > 0 && filename[bytes_read - 1] == '\n') {
        filename[bytes_read - 1] = '\0';
    } else {
        filename[bytes_read] = '\0';
    }

    int pipefd[2];
    if (pipe(pipefd) == -1) {
        const char error_msg[] = "Pipe creation failed\n";
        write(STDERR_FILENO, error_msg, sizeof(error_msg) - 1);
```

```
        exit(EXIT_FAILURE);
    }

    pid_t pid = fork();

    if (pid == -1) {
        const char error_msg[] = "Fork failed\n";
        write(STDERR_FILENO, error_msg, sizeof(error_msg) - 1);
        exit(EXIT_FAILURE);
    }

    if (pid == 0) {
        close(pipefd[0]);

        dup2(pipefd[1], STDOUT_FILENO);
        close(pipefd[1]);

        int file_fd = open(filename, O_RDONLY);
        if (file_fd == -1) {
            const char error_msg[] = "Error opening file\n";
            write(STDERR_FILENO, error_msg, sizeof(error_msg) - 1);
            exit(EXIT_FAILURE);
        }

        dup2(file_fd, STDIN_FILENO);
        close(file_fd);

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

        const char error_msg[] = "Exec failed\n";
        write(STDERR_FILENO, error_msg, sizeof(error_msg) - 1);
        exit(EXIT_FAILURE);

    } else {
        close(pipefd[1]);
```

```
        char buffer[1024];
        ssize_t bytes;

        while ((bytes = read(pipefd[0], buffer, sizeof(buffer))) > 0) {
            write(STDOUT_FILENO, buffer, bytes);
        }

        close(pipefd[0]);
        wait(NULL);
    }

    return EXIT_SUCCESS;
}
   wait(NULL);
}
```

child.c

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

int is_composite(int n) {
    if (n <= 3) return 0;
    if (n % 2 == 0 || n % 3 == 0) return 1;

    for (int i = 5; i * i <= n; i += 6) {
        if (n % i == 0 || n % (i + 2) == 0)
            return 1;
    }
    return 0;
}

int main() {
    char buffer[1024];
```

```
    ssize_t bytes_read;
    int number = 0;
    int is_negative = 0;
    int has_digits = 0;

    while ((bytes_read = read(STDIN_FILENO, buffer, sizeof(buffer))) > 0) {
        for (int i = 0; i < bytes_read; i++) {
            char c = buffer[i];
            if (c >= '0' && c <= '9') {
                number = number * 10 + (c - '0');
                has_digits = 1;
            } else if (c == '-' && !has_digits) {
                is_negative = 1;
            } else if (c == '\n' && has_digits) {

                if (is_negative) {
                    number = -number;
                }

                if (!is_composite(number)) {
                    exit(EXIT_SUCCESS);
                } else {
                    char output[32];
                    int len = 0;
                    int n = number;
                    char temp[32];
                    int temp_len = 0;
                    while (n > 0) {
                        temp[temp_len++] = '0' + (n % 10);
                        n /= 10;
                    }
                    for (int j = temp_len - 1; j >= 0; j--) {
                        output[len++] = temp[j];
                    }
                    output[len++] = '\n';
```

```
                write(STDOUT_FILENO, output, len);
            }

            number = 0;
            is_negative = 0;
            has_digits = 0;
        }
    }
}

if (has_digits) {
    if (is_negative) {
        number = -number;
    }

    if (!is_composite(number)) {
        exit(EXIT_SUCCESS);
    } else {
        char output[32];
        int len = 0;
        int n = number;
        char temp[32];
        int temp_len = 0;
        while (n > 0) {
            temp[temp_len++] = '0' + (n % 10);
            n /= 10;
        }
        for (int j = temp_len - 1; j >= 0; j--) {
            output[len++] = temp[j];
        }
        output[len++] = '\n';

        write(STDOUT_FILENO, output, len);
    }
```

```
    }


    return EXIT_SUCCESS;
}
```

Протокол работы программы

```
Enter filename: test1.txt
4
6
8
9
10
12
goida@zov:~/OS/lab1$ ./parent
Enter filename: test2.txt
6
10
14
goida@zov:~/OS/lab1$ ./parent
Enter filename: test3.txt
8
9
goida@zov:~/OS/lab1$
```

test1.txt:

```
1   4
2   6
3   8
4   9
5   10
6   12
```

test2.txt:

```
1   6
2   10
3   14
4   3
5
```

test3.txt:

```
1   8
2   9
3   -4
4   20
5   13
```

Вывод

В процессе выполнения лабораторной работы я составил программу, осуществляющую работу с процессами и взаимодействие между ними. Я приобрел базовые практические навыки в управлении процессами в ОС и обеспечении обмена между процессами посредством каналов. Одной из основных сложностей была в понимании принципа работы процессов и системных вызовов.