

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: Громова В.А.

Преподаватель: Бахарев В.Д.

Оценка: \_\_\_\_\_

Дата: 10.10.24

Москва, 2024

# Постановка задачи

## Вариант 7.

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 число число<newline>». Дочерний процесс считает их сумму и выводит результат в стандартный поток вывода. Числа имеют тип `float`. 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 односторонний канал между процессами, `fd[0]` для чтения, `fd[1]` для записи.
- `int open(const char *pathname, int flags)`; – открывает файл и возвращает файловый дескриптор.
- `ssize_t read(int fd, void *buf, size_t count)`; – читает данные из файла или канала в буфер.
- `ssize_t write(int fd, const void *buf, size_t count)`; – записывает данные в файл, канал или стандартный вывод.
- `int dup2(int oldfd, int newfd)`; – перенаправляет один файловый дескриптор на другой.
- `int execl(const char *path, const char *arg0, ..., NULL)`; – заменяет текущий процесс другим исполняемым файлом.
- `int wait(int *status)`; – родительский процесс ждет завершения дочернего процесса.
- `void _exit(int status)`; – завершает процесс без вызова функций очистки стандартной библиотеки.

Краткое описание работы программы.

Родительский процесс запрашивает у пользователя имя файла, открывает его для чтения и создаёт `pipe` для передачи данных дочернему процессу.

С помощью `fork()` создаётся дочерний процесс, который перенаправляет стандартный ввод на файл, а стандартный вывод – в `pipe`, и запускает программу `child`.

Программа `child` читает числа из файла, выполняет необходимые вычисления (суммирует числа или проверяет их на простоту) и записывает результаты в стандартный вывод, который направлен в `pipe`.

Родительский процесс читает данные из pipe и выводит их на экран. После завершения работы дочернего процесса родительский процесс корректно завершает выполнение программы.

## Код программы

parent.c

```
#include <unistd.h>

#include <stdlib.h>

#include <fcntl.h>

#include <sys/wait.h>


int main() {

    char filename[256];

    const char prompt[] = "введите имя файла: ";

    write(STDOUT_FILENO, prompt, sizeof(prompt) - 1);


    ssize_t nameLength = read(STDIN_FILENO, filename, sizeof(filename) - 1);

    if (nameLength <= 0) {

        const char err[] = "ошибка ввода имени файла\n";

        write(STDERR_FILENO, err, sizeof(err) - 1);

        _exit(EXIT_FAILURE);

    }


    if (filename[nameLength - 1] == '\n')

        filename[nameLength - 1] = '\0';

    else

        filename[nameLength] = '\0';


    int fileDescriptor = open(filename, O_RDONLY);

    if (fileDescriptor == -1) {

        const char err[] = "ошибка открытия файла\n";
```

```

        write(STDERR_FILENO, err, sizeof(err) - 1);

        _exit(EXIT_FAILURE);
    }

int pipefd[2];

if (pipe(pipefd) == -1) {

    const char err[] = "ошибка создания pipe\n";

    write(STDERR_FILENO, err, sizeof(err) - 1);

    _exit(EXIT_FAILURE);
}

pid_t childPid = fork();

if (childPid == -1) {

    const char err[] = "ошибка fork()\n";

    write(STDERR_FILENO, err, sizeof(err) - 1);

    _exit(EXIT_FAILURE);
}

if (childPid == 0) {

    close(pipefd[0]);

    dup2(fileDescriptor, STDIN_FILENO);

    dup2(pipefd[1], STDOUT_FILENO);

    close(fileDescriptor);

    close(pipefd[1]);

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

    const char err[] = "ошибка execl()\n";

    write(STDERR_FILENO, err, sizeof(err) - 1);

    _exit(EXIT_FAILURE);
} else {

    close(fileDescriptor);

    close(pipefd[1]);

```

```

    char buffer[256];

    ssize_t bytesRead;

    while ((bytesRead = read(pipefd[0], buffer, sizeof(buffer))) > 0) {

        write(STDOUT_FILENO, buffer, bytesRead);

    }

    close(pipefd[0]);

    wait(NULL);

}

return 0;
}

```

## child.c

```

#include <unistd.h>

#include <stdlib.h>

int main() {

    char buffer[256];

    ssize_t bytesRead;

    float totalSum;

    char *currentChar;

    while ((bytesRead = read(STDIN_FILENO, buffer, sizeof(buffer) - 1)) > 0) {

        buffer[bytesRead] = '\0';

        currentChar = buffer;

        totalSum = 0.0f;

        while (*currentChar) {

            while (*currentChar == ' ' || *currentChar == '\t' || *currentChar == '\n')

                currentChar++;

```

```
int sign = 1;

if (*currentChar == '-') {

    sign = -1;

    currentChar++;

}

float number = 0.0f;

float fractional = 0.1f;

int hasDigits = 0;

while (*currentChar >= '0' && *currentChar <= '9') {

    number = number * 10 + (*currentChar - '0');

    currentChar++;

    hasDigits = 1;

}

if (*currentChar == '.') {

    currentChar++;

    while (*currentChar >= '0' && *currentChar <= '9') {

        number += (*currentChar - '0') * fractional;

        fractional *= 0.1f;

        currentChar++;

        hasDigits = 1;

    }

}

if (hasDigits)

    totalSum += number * sign;
```

```
        while (*currentChar && *currentChar != ' ' && *currentChar != '\t' &&
*currentChar != '\n')

            currentChar++;

    }

    char output[64];

    int outIndex = 0;

    int integerPart = (int)totalSum;

    float fractionPart = totalSum - integerPart;

    if (fractionPart < 0) fractionPart = -fractionPart;

    char tmpDigits[32];

    int tmpIndex = 0;

    int absInteger = integerPart < 0 ? -integerPart : integerPart;

    do {

        tmpDigits[tmpIndex++] = '0' + (absInteger % 10);

        absInteger /= 10;

    } while (absInteger > 0);

    if (integerPart < 0)

        output[outIndex++] = '-';

    while (tmpIndex--)

        output[outIndex++] = tmpDigits[tmpIndex];

    output[outIndex++] = '.';

    for (int i = 0; i < 2; i++) {

        fractionPart *= 10;

        int digit = (int)fractionPart;
```

```
        output[outIndex++] = '0' + digit;

        fractionPart -= digit;

    }

    output[outIndex++] = '\n';

    write(STDOUT_FILENO, output, outIndex);

}

return 0;
}
```

```
}
```

## Вывод

Лабораторная работа показала, как использовать родительский и дочерний процессы с передачей данных через pipe. Программа корректно обрабатывает числа из файла и выводит результаты. Основная сложность заключалась в правильном перенаправлении потоков ввода/вывода через dup2. В будущем можно улучшить обработку ошибок и удобство ввода данных.