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: Авезов Р.А.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 03.10.25

Москва, 2025

Постановка задачи

Вариант 2.

Пользователь вводит команды вида: «число число число<endline>». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс считает их сумму и выводит её в файл. Числа имеют тип float. Количество чисел может быть произвольным.

Общий метод и алгоритм решения

Использованные системные вызовы:

1. pipe(int pipefd[2])

- Назначение: Создание односторонних каналов связи между родственными процессами.

2. fork(void)

- Назначение: Создание нового процесса-потомка, дублирующего адресное пространство родителя.

3. dup2 (int src_fd, int dst_fd)

- Назначение: Подмена стандартных потоков ввода-вывода на дескрипторы созданных каналов.

4. execl (const char path, const char arg, ...)

- Назначение: Загрузка нового исполняемого образа в контекст текущего процесса.

5. fgets(char buf, int size, FILE stream)

- Назначение: Буферизированное чтение текстовых строк из входного потока.

6. fprintf(FILE stream, const char fmt, ...)

- Назначение: Форматированный вывод данных в указанный поток.

7. open (const char name, const char mode)

- Назначение: Инициализация файлового объекта для работы с внешним файлом.

8. close(int fd)

- Назначение: Освобождение файлового дескриптора и связанных с ним ресурсов.

9. wait(pid_t pid)

- Назначение: Ожидание изменения состояния указанного процесса-потомка.

10. exit(int status)

- Назначение: Немедленное завершение процесса с передачей статуса родителю.

Разработанное приложение построено по архитектуре "ведущий-ведомый" с использованием двух направленных каналов. Основной процесс отвечает за диалог с пользователем - прием строк, содержащих числовые данные, и их передачу через первый

канал процессу-обработчику. Ведомый процесс выполняет синтаксический разбор полученных строк, идентифицирует вещественные числа, производит их суммирование и подсчет, сохраняя итоги в файл отчета. Через обратный канал передаются результирующие значения для информирования пользователя. Для обеспечения надежности реализована обработка исключительных ситуаций на всех этапах межпроцессного взаимодействия.

Код программы

child.c

```
#define _POSIX_C_SOURCE 200809L

#include <stdlib.h>

#include <string.h>

#include <errno.h>

#include <unistd.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdio.h>


static int SumFloatsInLine(const char *line, float *outSum, int *outCount) {

    if (!line || !outSum || !outCount) return -1;

    float sum = 0.0f;

    int count = 0;


    char buf[1024];

    if (strlen(line) >= sizeof(buf)) return -1;

    strcpy(buf, line);


    char *token = strtok(buf, " \t");

    while (token) {

        char *endptr = NULL;

        errno = 0;

        float val = strtof(token, &endptr);
```

```
        if (endptr == token || *endptr != '\0') {
        } else if (errno == ERANGE) {
        } else {
            sum += val;
            count++;
        }
        token = strtok(NULL, " \t");
    }
    *outSum = sum;
    *outCount = count;
    return 0;
}

int main(int argc, char *argv[]) {
    if (argc < 2) {
        char usageMsg[100];
        snprintf(usageMsg, sizeof(usageMsg), "Использование: %s <output_file>\n", argv[0]);
        write(STDERR_FILENO, usageMsg, strlen(usageMsg));
        return 1;
    }

    const char *outPath = argv[1];
    int fout = open(outPath, O_WRONLY | O_CREAT | O_APPEND, 0644);
    if (fout == -1) {
        char errorMsg[] = "fopen output: ";
        write(STDERR_FILENO, errorMsg, sizeof(errorMsg) - 1);
        perror("");
        return 1;
    }
```

```
char line[1024];
char resultMsg[512];
char errorMsg[100];

while (1) {
    ssize_t bytesRead = read(STDIN_FILENO, line, sizeof(line) - 1);
    if (bytesRead <= 0) {
        if (bytesRead < 0) {
            char readError[] = "fgets stdin: ";
            write(STDERR_FILENO, readError, sizeof(readError) - 1);
            perror("");
        }
        break;
    }
    line[bytesRead] = '\0';

    size_t l = strlen(line);
    if (l > 0 && line[l-1] == '\n') line[l-1] = '\0';

    float sum = 0.0f;
    int count = 0;

    if (SumFloatsInLine(line, &sum, &count) != 0) {
        char parseError[] = "error: parse line\n";
        write(STDERR_FILENO, parseError, sizeof(parseError) - 1);
        continue;
    }

    if (count == 0) {
        char noNumbers[] = "no numbers\n";
```

```
            write(STDOUT_FILENO, noNumbers, sizeof(noNumbers) - 1);
            continue;
        }

        char fileLine[1024];
        snprintf(fileLine, sizeof(fileLine), "line: \"%s\" sum: %.6f count: %d\n", line, sum, count);
        ssize_t written = write(fout, fileLine, strlen(fileLine));
        if (written <= 0) {
            char writeError[] = "error: write to file\n";
            write(STDERR_FILENO, writeError, sizeof(writeError) - 1);
        }

        snprintf(resultMsg, sizeof(resultMsg), "sum=%.6f count=%d\n", sum, count);
        write(STDOUT_FILENO, resultMsg, strlen(resultMsg));
    }

    close(fout);
    return 0;
}
```

parent.c

```
#define _POSIX_C_SOURCE 200809L
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/stat.h>
#include <fcntl.h>
```

```
#include <stdio.h>

int main() {
    char fileName[256];
    char prompt[] = "Введите имя файла для результатов (например, results.txt): ";
    write(STDOUT_FILENO, prompt, sizeof(prompt) - 1);

    ssize_t bytesRead = read(STDIN_FILENO, fileName, sizeof(fileName) - 1);
    if (bytesRead <= 0) {
        char errorMsg[] = "Ошибка: не удалось прочитать имя файла\n";
        write(STDERR_FILENO, errorMsg, sizeof(errorMsg) - 1);
        return 1;
    }
    fileName[bytesRead] = '\0';

    size_t len = strlen(fileName);
    if (len > 0 && fileName[len - 1] == '\n') {
        fileName[len - 1] = '\0';
    }

    if (strlen(fileName) == 0) {
        char errorMsg[] = "Ошибка: имя файла пустое\n";
        write(STDERR_FILENO, errorMsg, sizeof(errorMsg) - 1);
        return 1;
    }

    int pipe1[2];
    int pipe2[2];

    if (pipe(pipe1) == -1) {
```

```
    char errorMsg[] = "pipe1: ";
    write(STDERR_FILENO, errorMsg, sizeof(errorMsg) - 1);
    perror("");
    return 1;
}

if (pipe(pipe2) == -1) {
    char errorMsg[] = "pipe2: ";
    write(STDERR_FILENO, errorMsg, sizeof(errorMsg) - 1);
    perror("");
    close(pipe1[0]);
    close(pipe1[1]);
    return 1;
}

pid_t pid = fork();
if (pid < 0) {
    char errorMsg[] = "fork: ";
    write(STDERR_FILENO, errorMsg, sizeof(errorMsg) - 1);
    perror("");
    close(pipe1[0]);
    close(pipe1[1]);
    close(pipe2[0]);
    close(pipe2[1]);
    return 1;
}

if (pid == 0) {
    close(pipe1[1]);
    close(pipe2[0]);
```

```
        if (dup2(pipe1[0], STDIN_FILENO) == -1) {
            perror("dup2 stdin");
            _exit(1);
        }
        if (dup2(pipe2[1], STDOUT_FILENO) == -1) {
            perror("dup2 stdout");
            _exit(1);
        }

        close(pipe1[0]);
        close(pipe2[1]);

        execl("./child", "child", fileName, (char*)NULL);
        perror("execl child");
        _exit(1);
    }

    close(pipe1[0]);
    close(pipe2[1]);

    char infoMsg1[] = "Теперь вводите строки с числами (float через точку), например: 1.5 2 3\n";
    char infoMsg2[] = "Для завершения введите: exit\n";
    write(STDOUT_FILENO, infoMsg1, sizeof(infoMsg1) - 1);
    write(STDOUT_FILENO, infoMsg2, sizeof(infoMsg2) - 1);

    char line[1024];
    char childReply[512];
```

```
while (1) {
    char promptSymbol[] = "> ";
    write(STDOUT_FILENO, promptSymbol, sizeof(promptSymbol) - 1);

    bytesRead = read(STDIN_FILENO, line, sizeof(line) - 1);
    if (bytesRead <= 0) {
        if (bytesRead == 0) {
            char eofMsg[] = "\nКонец ввода\n";
            write(STDOUT_FILENO, eofMsg, sizeof(eofMsg) - 1);
        } else {
            perror("read stdin");
        }
        break;
    }
    line[bytesRead] = '\0';

    size_t l = strlen(line);
    if (l > 0 && line[l - 1] == '\n') {
        line[l - 1] = '\0';
    }

    if (strcmp(line, "exit") == 0) {
        break;
    }

    char lineWithNewline[1024];
    snprintf(lineWithNewline, sizeof(lineWithNewline), "%s\n", line);
    ssize_t written = write(pipe1[1], lineWithNewline, strlen(lineWithNewline));
    if (written <= 0) {
        perror("write to child");
```

```
            break;
        }

        bytesRead = read(pipe2[0], childReply, sizeof(childReply) - 1);
        if (bytesRead > 0) {
            childReply[bytesRead] = '\0';
            char prefix[] = "[child] ";
            write(STDOUT_FILENO, prefix, sizeof(prefix) - 1);
            write(STDOUT_FILENO, childReply, bytesRead);
        } else {
            if (bytesRead == 0) {
                char closedMsg[] = "Дочерний процесс закрыл канал\n";
                write(STDOUT_FILENO, closedMsg, sizeof(closedMsg) - 1);
            } else {
                perror("read from child");
            }
            break;
        }
    }

    close(pipe1[1]);
    close(pipe2[0]);

    int status = 0;
    if (waitpid(pid, &status, 0) == -1) {
        perror("waitpid");
        return 1;
    }

    if (WIFEXITED(status)) {
```

```
        int code = WEXITSTATUS(status);
        char doneMsg[100];
        if (code == 0) {
            snprintf(doneMsg, sizeof(doneMsg), "Готово. Дочерний процесс завершился успешно.\n");
        } else {
            snprintf(doneMsg, sizeof(doneMsg), "Дочерний процесс завершился с кодом %d.\n", code);
        }
        write(STDOUT_FILENO, doneMsg, strlen(doneMsg));
    } else if (WIFSIGNALED(status)) {
        char signalMsg[100];
        snprintf(signalMsg, sizeof(signalMsg), "Дочерний процесс завершён сигналом %d.\n", WTERMSIG(status));
        write(STDOUT_FILENO, signalMsg, strlen(signalMsg));
    }

    return 0;
}
```

Протокол работы программы

```
rustam@MacBook-Air-Rustam build % ./parent
Введите имя файла для результатов (например, results.txt): results.txt
Теперь вводите строки с числами (float через точку), например: 1.5 2 3
Для завершения введите: exit
> 2.6 4 3
[child] sum=9.600000 count=3
> 10 20 30
[child] sum=60.000000 count=3
> 20 3.4 4.5
[child] sum=27.900000 count=3
> exit
Готово. Дочерний процесс завершился успешно.
```

Вывод

В ходе лабораторной работы была разработана программа, организующая взаимодействие между родительским и дочерним процессами через каналы (pipes). Программа успешно обрабатывает вводимые пользователем строки, извлекает из них числа с плавающей точкой, вычисляет их сумму и сохраняет результаты в файл. Все требования задания выполнены -

реализовано управление процессами, межпроцессное взаимодействие и обработка системных ошибок. Работа продемонстрировала практическое применение системных вызовов POSIX для создания многопроцессных приложений в Unix-подобных операционных системах. Программа корректно выполняет возложенные на нее функции и надежно завершает работу при любом сценарии использования.