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 <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <errno.h>

#include <unistd.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) {
        fprintf(stderr, "Использование: %s <output_file>\n", argv[0]);
        return 1;
    }

    const char *outPath = argv[1];
    FILE *fout = fopen(outPath, "a");
    if (!fout) {
        perror("fopen output");
        return 1;
    }

    char line[1024];
    while (fgets(line, sizeof(line), stdin)) {
        if (ferror(stdin)) {
            perror("fgets stdin");
            break;
        }
```

```
        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) {
            fprintf(stderr, "error: parse line\n");
            continue;
        }

        if (count == 0) {
            printf("no numbers\n");
            fflush(stdout);
            continue;
        }

        if (fprintf(fout, "line: \"%s\" sum: %.6f count: %d\n", line, sum, count) < 0) {
            fprintf(stderr, "error: write to file\n");
        }
        fflush(fout);

        printf("sum=%.6f count=%d\n", sum, count);
        fflush(stdout);
    }

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

parent.c

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

int main() {
    char fileName[256];
    printf("Введите имя файла для результатов (например, results.txt): ");
    fflush(stdout);

    if (!fgets(fileName, sizeof(fileName), stdin)) {
        fprintf(stderr, "Ошибка: не удалось прочитать имя файла\n");
        return 1;
    }

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

    if (strlen(fileName) == 0) {
        fprintf(stderr, "Ошибка: имя файла пустое\n");
        return 1;
    }
```

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

if (pipe(pipe1) == -1) {
    perror("pipe1");
    return 1;
}

if (pipe(pipe2) == -1) {
    perror("pipe2");
    close(pipe1[0]);
    close(pipe1[1]);
    return 1;
}

pid_t pid = fork();
if (pid < 0) {
    perror("fork");
    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]);

FILE *toChild = fdopen(pipe1[1], "w");
if (!toChild) {
    perror("fdopen toChild");
    close(pipe1[1]);
    close(pipe2[0]);
    kill(pid, SIGTERM);
    waitpid(pid, NULL, 0);
    return 1;
}

FILE *fromChild = fdopen(pipe2[0], "r");
if (!fromChild) {
```

```
    perror("fdopen fromChild");
    fclose(toChild);
    close(pipe2[0]);
    kill(pid, SIGTERM);
    waitpid(pid, NULL, 0);
    return 1;
}

printf("Теперь вводите строки с числами (float через точку), например: 1.5 2 3\n");
printf("Для завершения введите: exit\n");

char line[1024];
while (1) {
    printf("> ");
    fflush(stdout);

    if (!fgets(line, sizeof(line), stdin)) {
        if (feof(stdin)) {
            printf("\nКонец ввода\n");
        } else {
            perror("fgets stdin");
        }
        break;
    }

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

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

```
    }

    if (fprintf(toChild, "%s\n", line) < 0) {
        perror("fprintf toChild");
        break;
    }
    fflush(toChild);

    char reply[512];
    if (fgets(reply, sizeof(reply), fromChild)) {
        size_t rl = strlen(reply);
        if (rl > 0 && reply[rl - 1] == '\n') reply[rl - 1] = '\0';
        printf("[child] %s\n", reply);
    } else {
        if (feof(fromChild)) {
            printf("Дочерний процесс закрыл канал\n");
        } else {
            perror("fgets fromChild");
        }
        break;
    }
}

fclose(toChild);
fclose(fromChild);

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

```
    }

    if (WIFEXITED(status)) {
        int code = WEXITSTATUS(status);
        if (code == 0) {
            printf("Готово. Дочерний процесс завершился успешно.\n");
        } else {
            printf("Дочерний процесс завершился с кодом %d.\n", code);
        }
    } else if (WIFSIGNALED(status)) {
        printf("Дочерний процесс завершён сигналом %d.\n", WTERMSIG(status));
    } else if (WIFSTOPPED(status)) {
        printf("Дочерний процесс остановлен сигналом %d.\n", WSTOPSIG(status));
    }

    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.