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-209БВ-24

Студент: Фимина А.О.

Преподаватель: Миронов Е.С.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: 08.10.25

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.

**Общий метод и алгоритм решения**

Использованные системные вызовы:

• pid\_t fork(void) - создает дочерний процесс

• int pipe(int \*fd) - создает канал и возвращает два дескриптора

• int close(int fd) - закрывает файловый дескриптор

• int dup2(int oldfd, int newfd) - дублирует файловый дескриптор

• int execv(const char \*path, char \*const argv[]) - загружает и запускает новую программу

• pid\_t waitpid(pid\_t pid, int \*status, int options) - ожидает завершения дочернего процесса

• ssize\_t write(int fd, const void \*buf, size\_t count) - записывает данные в файловый дескриптор

• ssize\_t read(int fd, void \*buf, size\_t count) - читает данные из файлового дескриптора

Алгоритм работы:

1. Родительский процесс запрашивает у пользователя имя выходного файла

2. Создаются два pipe'а для связи между процессами

3. Создается дочерний процесс с помощью fork()

4. В дочернем процессе:

- Закрываются ненужные концы pipe'ов

- Перенаправляется stdin на чтение из pipe1

- Перенаправляется stdout на запись в pipe2

- Запускается программа child с помощью execv()

5. В родительском процессе:

- Закрываются ненужные концы pipe'ов

- Считываются строки с числами от пользователя

- Строки передаются дочернему процессу через pipe1

- Результаты работы дочернего процесса читаются из pipe2

- Ожидается завершение дочернего процесса

**Код программы**

**parent.cpp**

#include <iostream>

#include <string>

#include <unistd.h>

#include <sys/wait.h>

#include <vector>

#include <cstring>

int main() {

std::string filename;

std::cout << "Enter output filename: ";

std::getline(std::cin, filename);

if (filename.empty()) {

std::cerr << "Filename cannot be empty!\n";

return 1;

}

// Создаём два pipe'а

int pipe1[2]; // parent -> child (stdin)

int pipe2[2]; // child -> parent (stdout) — опционально

if (pipe(pipe1) == -1 || pipe(pipe2) == -1) {

perror("pipe");

return 1;

}

pid\_t pid = fork();

if (pid == -1) {

perror("fork");

return 1;

}

if (pid == 0) {

// Дочерний процесс

// Закрываем ненужные концы pipe'ов

close(pipe1[1]); // закрываем write-end pipe1

close(pipe2[0]); // закрываем read-end pipe2

// Перенаправляем stdin из pipe1[0]

dup2(pipe1[0], STDIN\_FILENO);

close(pipe1[0]);

// Перенаправляем stdout в pipe2[1] (опционально)

dup2(pipe2[1], STDOUT\_FILENO);

close(pipe2[1]);

// Подготавливаем аргументы для execv

char\* child\_argv[] = {

const\_cast<char\*>("./child"),

const\_cast<char\*>(filename.c\_str()),

nullptr

};

execv("./child", child\_argv);

perror("execv failed");

exit(1);

} else {

// Родительский процесс

close(pipe1[0]); // не читаем из pipe1

close(pipe2[1]); // не пишем в pipe2

std::string input\_line;

std::cout << "Enter lines of floats (Ctrl+D to finish):\n";

while (std::getline(std::cin, input\_line)) {

if (input\_line.empty()) continue;

input\_line += '\n'; // getline убирает \n

write(pipe1[1], input\_line.c\_str(), input\_line.size());

}

// Закрываем write-end: это сигнализирует ребёнку об окончании ввода

close(pipe1[1]);

// (Опционально) читаем ответ от ребёнка

char buffer[256];

ssize\_t bytes;

std::cout << "Child output (if any):\n";

while ((bytes = read(pipe2[0], buffer, sizeof(buffer) - 1)) > 0) {

buffer[bytes] = '\0';

std::cout << buffer;

}

close(pipe2[0]);

// Ждём завершения дочернего процесса

int status;

waitpid(pid, &status, 0);

if (WIFEXITED(status)) {

std::cout << "Child exited with status " << WEXITSTATUS(status) << "\n";

}

}

return 0;

}

**child.cpp**

#include <iostream>

#include <fstream>

#include <sstream>

#include <string>

#include <cstdlib>

int main(int argc, char\* argv[]) {

if (argc != 2) {

std::cerr << "Usage: " << argv[0] << " <output\_file>\n";

return 1;

}

std::string output\_file = argv[1];

std::ofstream out(output\_file);

if (!out.is\_open()) {

std::cerr << "Cannot open output file: " << output\_file << "\n";

return 1;

}

std::string line;

while (std::getline(std::cin, line)) {

if (line.empty()) continue;

std::istringstream iss(line);

float num, sum = 0.0f;

int count = 0;

while (iss >> num) {

sum += num;

++count;

}

if (count > 0) {

out << "Sum: " << sum << "\n";

out.flush(); // чтобы сразу записалось

}

}

out.close();

return 0;

}

**Протокол работы программы**

Тестирование:

$ ./parent

Enter output filename: result.txt

Enter lines of floats (Ctrl+D to finish):

1 2 3 1

-0.5

^D

Child output (if any):

Child exited with status 0

$ cat result.txt

Sum: 7

Sum: -0.5

Strace(через docker):

$ strace -f ./parent

execve("./parent", ["./parent"], 0x7ff... /\* env \*/) = 0 <- запуск процесса

pipe([3, 4]) = 0 <- pipe

pipe([5, 6]) = 0 <- pipe

fork() = 17062 <- fork

[pid 17062] close(4) = 0; close(5) = 0

[pid 17062] dup2(3, 0) = 0 <- dup2(stdin)

[pid 17062] dup2(6, 1) = 1 <- dup2(stdout)

[pid 17062] execv("./child", ["./child", "result.txt", NULL]) = 0 <- execv

write(3, "1 2 3 1\n", 8) = 8 <- write к ребёнку

write(3, "-0.5\n", 5) = 5 <- write к ребёнку

close(3) = 0 <- закрытие записи (EOF ребёнку)

waitpid(17062, [WIFEXITED(s) && WEXITSTATUS(s)=0], 0) = 17062 <- waitpid

--- exited with 0 ---

**Вывод**

В ходе выполнения лабораторной работы я изучила:

- Создание и использование pipe'ов для межпроцессного взаимодействия

- Работу с системными вызовами fork(), execv(), dup2()

- Перенаправление стандартных потоков ввода/вывода

- Обработку завершения дочерних процессов с помощью waitpid()

- Передачу данных между процессами через каналы

Программа успешно реализует требуемую функциональность: родительский процесс передает строки с числами дочернему процессу, который вычисляет сумму чисел в каждой строке и записывает результат в указанный файл. Использование pipe'ов обеспечивает надежную передачу данных между процессами.