

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-211БВ-24

Студент: ПРОХОРОВ В.И.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 15.10.25

Москва, 2025

Постановка задачи

Вариант 2.

Пользователь вводит команды вида: «число число число<newline>». 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:

- `pid_t fork(void)`; – создает дочерний процесс.
- `int pipe(int fd[2])`; – создаёт канал связи (pipe), возвращает 0 при успехе, -1 при ошибке
- `ssize_t read(int fd, void *buf, size_t count)` - читает данные из файла по дескриптору fd в буфер buf. Возвращает количество прочитанных байт
- `ssize_t write(int fd, const void *buf, size_t count)` - записывает данные в файл по дескриптору fd. Возвращает число записанных байт
- `int open(const char *pathname, int flag, mode_t mode)` - Открывает файл, возвращает файловый дескриптор
- `int close(int fd)` - закрывает файловый дескриптор
- `int execv(const char *path, char *const argv[])` - заменяет текущий процесс новым. Возвращает -1 при ошибке
- `int dup2(int fd, int fd2)` - замена файлового дескриптора
- `pid_t wait(int *wstatus)` - ожидает завершения дочернего процесса. Возвращает pid завершившегося процесса
- `ssize_t readlink(const char *path, char *buf, size_t size)` - определяет путь к текущему исполняемому файлу
- `pid_t getpid(void)` - возвращает pid текущего процесса
- `ssize_t readlink(const char *pathname, char *buf, size_t bufsiz)` - считывает путь, на который указывает символьная ссылка

Клиент и сервер общаются через pipe. Родитель запускает дочерний процесс, перенаправляя ему стандартный ввод и вывод на pipe. Из консоли дочерний процесс читает числа, отправляет их родителю. Родитель принимает строку чисел из stdin, читает её и складывает полученные из нее числа в результат. Результаты сервер записывает в выходной файл.

Код программы

parent.c

```
#include <stdint.h>
#include <stdbool.h>

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

```

static char CHILD_PROGRAM_NAME[] = "child";

int main(int argc, char *argv[]) {
    if (argc == 1) {
        char msg[256];
        uint32_t len = snprintf(msg, sizeof(msg) - 1, "usage: %s filename\n", argv[0]); //
        // длина строки
        write(STDERR_FILENO, msg, len);
        exit(EXIT_SUCCESS);
    }

    // Получение пути
    char progbath[2048];
    ssize_t len = readlink("/proc/self/exe", progbath, sizeof(progbath) - 1);
    if (len == -1) {
        const char msg[] = "Failed to read /proc/self/exe\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }
    while (progbath[len] != '/')
        --len;
    progbath[len] = '\0';

    // Создание pipe
    int parent_to_child[2];
    int child_to_parent[2];
    if (pipe(parent_to_child) == -1 || pipe(child_to_parent) == -1) {
        const char msg[] = "Failed to create pipes\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }

    pid_t pid = fork();
    if (pid == -1) {
        const char msg[] = "Failed to fork\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }

    if (pid == 0) { // Дочерний процесс
        close(parent_to_child[1]);
        close(child_to_parent[0]);

        dup2(parent_to_child[0], STDIN_FILENO);
        dup2(child_to_parent[1], STDOUT_FILENO);

        close(parent_to_child[0]);
        close(child_to_parent[1]);

        char path[4096];
        snprintf(path, sizeof(path) - 1, "%s/%s", progbath, CHILD_PROGRAM_NAME);

        char *const args[] = {CHILD_PROGRAM_NAME, argv[1], NULL};
    }
}

```

```

    execv(path, args);

    const char msg[] = "Failed to exec child\n";
    write(STDERR_FILENO, msg, sizeof(msg));
    exit(EXIT_FAILURE);
}

close(parent_to_child[0]);
close(child_to_parent[1]);

char input_buf[1024];
ssize_t bytes = read(STDIN_FILENO, input_buf, sizeof(input_buf));
if (bytes < 0) {
    const char msg[] = "Failed to read from stdin\n";
    write(STDERR_FILENO, msg, sizeof(msg));
    exit(EXIT_FAILURE);
}

write(parent_to_child[1], input_buf, bytes);
close(parent_to_child[1]); // EOF для ребёнка

char result_buf[256];
ssize_t result_bytes = read(child_to_parent[0], result_buf, sizeof(result_buf));
if (result_bytes > 0) {
    write(STDOUT_FILENO, result_buf, result_bytes);
} else {
    const char msg[] = "No result from child\n";
    write(STDERR_FILENO, msg, sizeof(msg));
}

close(child_to_parent[0]);

int status;
wait(&status);
if (WIFEXITED(status) && WEXITSTATUS(status) == 0) {
    exit(EXIT_SUCCESS);
} else {
    exit(EXIT_FAILURE);
}
}

```

child.c

```

#include <stdint.h>
#include <stdbool.h>

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

int main(int argc, char **argv) {

```

```

if (argc != 2) {
    const char msg[] = "ERROR: filename argument missing\n";
    write(STDOUT_FILENO, msg, sizeof(msg) - 1);
    exit(EXIT_FAILURE);
}

const char *filename = argv[1];
int fd = open(filename, O_WRONLY | O_CREAT | O_TRUNC, 0644);
if (fd == -1) {
    const char msg[] = "ERROR: cannot open file\n";
    write(STDOUT_FILENO, msg, sizeof(msg) - 1);
    exit(EXIT_FAILURE);
}

char buf[1024];
ssize_t bytes = read(STDIN_FILENO, buf, sizeof(buf) - 1);
if (bytes <= 0) {
    const char msg[] = "ERROR: no input\n";
    write(STDOUT_FILENO, msg, sizeof(msg) - 1);
    close(fd);
    exit(EXIT_FAILURE);
}
buf[bytes] = '\0';

// парсим строку
float sum = 0.0;
char *p = buf;
while (*p) {
    while (*p && (*p == ' ' || *p == '\n' || *p == '\t'))
        p++;
    if (!*p) break;

    char *end;
    float val = strtod(p, &end);
    if (p == end) break;
    sum += val;
    p = end;
}

char result[128];
int len = snprintf(result, sizeof(result), "%.3f\n", sum);

write(fd, result, len);          // в файл
write(STDOUT_FILENO, result, len); // родителю
close(fd);

exit(EXIT_SUCCESS);
}

```

Протокол работы программы

```
proho@Persona21:~/prog/os_labs/lab_1$ ./parent out.txt
1. 2. 3. -1.21
4.790
```

```
lab_1 > out.txt
1 4.790
2
```

```
proho@Persona21:~/prog/os_labs/lab_1$ ./parent out.txt
1.2 2.1 3.123 2. -1.2
7.223
```

```
lab_1 > out.txt
1 7.223
2
```

Вывод

При выполнении данной лабораторной работы я научился работать с процессами в ОС. Попробовал создавать дочерние процессы и устанавливать каналы связи с помощью `fork`, `pipe` и других системных вызовов. Научился обрабатывать ввод, вывод как поток байтов, преобразовывать его в строки.