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-214Б-23

Студент: Караткевич Н. С.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 16.02.25

Москва, 2025

Постановка задачи

Вариант 7.

В файле записаны команды вида: «число число число<endline>». 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 execve(const char *filename, char *const argv[], char *const envp[]) (и другие вариации
- exec) - замена образа памяти процесса
- pid_t wait() - Ожидание завершения дочерних процессов
- int pipe(int pipefd[2]) - создание неименованного канала для передачи данных между процессами
- int dup2(int oldfd, int newfd) - переназначение файлового дескриптора
- int open(const char *pathname, int flags, mode_t mode) - открытие\создание файла
- int close(int fd) - закрыть файл ·

Описание работы программы

Программа состоит из родительского (main) и дочернего (child) процессов. Родительский процесс создает дочерний процесс, который выполняет вычисления и передает результат обратно через пайп.

Алгоритм работы main.c (родительский процесс):

1. **Открытие файла:**
   - **open(argv[1], O_RDONLY); – открываем файл, переданный через аргументы командной строки.**
   - **Если файл не удалось открыть, программа завершает работу с ошибкой.**
2. **Создание пайпа:**

- **pipe(pipefd); – создаем канал для связи родительского и дочернего процессов.**
- **pipefd[0] – используется для чтения, pipefd[1] – для записи.**

3. **Создание дочернего процесса:**
   - **fork(); – создаем новый процесс.**
   - **В дочернем процессе перенаправляем ввод и вывод:**
     - **dup2(input, STDIN_FILENO); – перенаправляем ввод с файла.**
     - **dup2(pipefd[1], STDOUT_FILENO); – перенаправляем вывод в пайп.**
     - **close(pipefd[0]); – закрываем ненужные дескрипторы.**
     - **execl("./child.out", "child.out", NULL); – запускаем child.out.**
4. **Ожидание завершения дочернего процесса:**
   - **wait(NULL); – ждем завершения child.out.**
5. **Чтение данных из пайпа:**
   - **Читаем результат, переданный child.out, и выводим на экран.**
6. **Закрытие пайпа:**
   - **close(pipefd[0]); – закрываем дескриптор чтения.**

Алгоритм работы child.c (дочерний процесс):

1. **Чтение чисел из stdin (который теперь связан с файлом).**
2. **Разбор строки на числа (float) и их суммирование.**
3. **Вывод суммы в stdout, который перенаправлен в пайп.**
4. **Повтор для каждой строки: сумма сбрасывается, и начинается новый цикл.**
5. **Завершение работы: return 0;.**

Взаимодействие процессов

- **Родительский процесс открывает файл и передает его дочернему через dup2().**
- **Дочерний процесс читает числа, суммирует их и отправляет результат родителю через пайп.**
- **Родительский процесс читает сумму из пайпа и выводит на экран.**

Основной поток работы:

1. **Родитель → Файл (открывает файл).**
2. **Родитель → Дочерний (создает через fork()).**

3. **Дочерний → Файл (читает данные).**
4. **Дочерний → Пайп (передает результат).**
5. **Родитель ← Пайп (читает сумму и выводит).**
6. **Родитель → Завершение (wait()).**

Код программы

main.c (**Родительский процесс**)
Этот файл **создаёт дочерний процесс** и перенаправляет ввод/вывод с помощью пайпа.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    if (argc < 2) {
        write(STDERR_FILENO, "Ошибка: укажите файл с числами\n", 32);
        return 1;
    }

    int input = open(argv[1], O_RDONLY);
    if (input < 0) {
        perror("Ошибка открытия файла");
        return 1;
    }

    int pipefd[2];
    if (pipe(pipefd) == -1) {
        perror("Ошибка создания пайпа");
        return 1;
    }

    pid_t child = fork();
    if (child == -1) {
        perror("Ошибка fork");
        return 1;
    }

    if (child == 0) {  // Дочерний процесс
        dup2(input, STDIN_FILENO);      // Входной поток теперь из файла
        dup2(pipefd[1], STDOUT_FILENO); // Выходной поток теперь в пайп
        close(pipefd[0]);  // Закрываем чтение из пайпа в дочернем процессе
        execl("./child.out", "child.out", NULL);
```

```
        perror("Ошибка exec");
        exit(1);
    } else {  // Родительский процесс
        close(pipefd[1]); // Закрываем запись в пайп, будем только читать

        wait(NULL); // Ожидаем завершения дочернего процесса

        char buffer[128];
        int n;
        while ((n = read(pipefd[0], buffer, sizeof(buffer))) > 0) {
            write(STDOUT_FILENO, buffer, n);
        }
        close(pipefd[0]);
    }
    return 0;
}
```

child.c (Дочерний процесс)
Этот файл **читает числа, суммирует их и отправляет результат** в stdout.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

int main() {
    char buffer[128];
    float sum = 0.0f;

    while (fgets(buffer, sizeof(buffer), stdin)) {
        char *ptr = buffer;
        while (*ptr) {
            float num = strtof(ptr, &ptr); // Преобразуем строку в float
            sum += num;
            while (*ptr == ' ') ptr++; // Пропускаем пробелы
        }
        printf("%.2f\n", sum);
        fflush(stdout); // Выводим результат сразу
        sum = 0.0f;
    }
    return 0;
}
```

func.h (Заголовочный файл)

```
#ifndef __FUNC_H__
#define __FUNC_H__

#include <stddef.h>

char *strnchr(const char *buf, char c, size_t len);
int read_line(int fd, char **buf, int *buf_size);
```

```
int print(int fd, const char *s);

#endif
```

---

func.c (Дополнительные функции)

```
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

char *strnchr(const char *buf, char c, size_t len) {
    for (size_t i = 0; i < len; i++) {
        if (buf[i] == c)
            return (char *)(buf + i);
        if (buf[i] == 0)
            return 0;
    }
    return 0;
}

int read_line(int fd, char **buf, int *buf_size) {
    int count = strnchr(*buf, 0, *buf_size) - *buf + 1;
    for (int i = 0; i < *buf_size - count; i++) {
        (*buf)[i] = (*buf)[i + count];
    }
    count = strnchr(*buf, 0, *buf_size) - *buf;
    int read_cur;
    do {
        if (count + 1 >= *buf_size) {
            int new_size = *buf_size * 2;
            char *tmp = realloc(*buf, new_size);
            if (!tmp)
                return count;
            *buf = tmp;
            *buf_size = new_size;
        }
        read_cur = read(fd, *buf + count, *buf_size - count - 1);
        count += read_cur;
        (*buf)[count] = 0;
    } while (read_cur > 0 && !strnchr(*buf, '\n', *buf_size));
    int line_len = strnchr(*buf, '\n', *buf_size) - *buf;
    if (line_len < 0)
        return count;
    (*buf)[line_len] = 0;
    return line_len + 1;
}

int print(int fd, const char *s) {
    int n = strlen(s);
    return write(fd, s, n);
}
```

```
gcc -o main.out main.c func.c
gcc -o child.out child.c
./main.out inp.txt
```

Протокол работы программы

./main.out

inp.txt

1.00

30.87

8.00

6.60

Тестирование:

```
$ strace -f ./solution.out inp.txt
execve("./solution.out", ["./solution.out", "inp.txt"], 0x7ffd9c33faf0 /* 65 vars */) = 0
brk(NULL)                               = 0x5636b8fbf000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffdaaf35f90) = -1 EINVAL (Недопустимый аргумент)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fc9ac35a000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (Нет такого файла или каталога)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=82523, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 82523, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fc9ac345000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\0\0\0\0\0"..., 832) = 832
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) =
784
pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48, 848) =
48
```

```
pread64(3,
"\4\0\0\0\24\0\0\0\3\0\0\0GNU\0I\17\357\204\3$\f\221\2039x\324\224\323\236S"..., 68, 896)
= 68
newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, AT_EMPTY_PATH) = 0
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) =
784
mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fc9ac11c000
mprotect(0x7fc9ac144000, 2023424, PROT_NONE) = 0
mmap(0x7fc9ac144000, 1658880, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7fc9ac144000
mmap(0x7fc9ac2d9000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1bd000) = 0x7fc9ac2d9000
mmap(0x7fc9ac332000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x7fc9ac332000
mmap(0x7fc9ac338000, 52816, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fc9ac338000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fc9ac119000
arch_prctl(ARCH_SET_FS, 0x7fc9ac119740) = 0
set_tid_address(0x7fc9ac119a10)         = 15625
set_robust_list(0x7fc9ac119a20, 24)     = 0
rseq(0x7fc9ac11a0e0, 0x20, 0, 0x53053053) = 0
mprotect(0x7fc9ac332000, 16384, PROT_READ) = 0
mprotect(0x5636b8e32000, 4096, PROT_READ) = 0
mprotect(0x7fc9ac394000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7fc9ac345000, 82523)           = 0
openat(AT_FDCWD, "inp.txt", O_RDONLY)   = 3
pipe2([4, 5], 0)                        = 0
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7fc9ac119a10) = 15626
dup2(4, 0)                              = 0
close(5strace: Process 15626 attached
)                                       = 0
[pid 15625] wait4(-1,  <unfinished ...>
[pid 15626] set_robust_list(0x7fc9ac119a20, 24) = 0
[pid 15626] dup2(3, 0)                  = 0
[pid 15626] dup2(5, 1)                  = 1
[pid 15626] close(4)                    = 0
[pid 15626] execve("child.out", ["./solution.out", "inp.txt"], NULL) = 0
[pid 15626] brk(NULL)                   = 0x55c27b074000
[pid 15626] arch_prctl(0x3001 /* ARCH_??? */, 0x7ffdfee75590) = -1 EINVAL (Недопустимый
аргумент)
[pid 15626] mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
0) = 0x7f6b27543000
[pid 15626] access("/etc/ld.so.preload", R_OK) = -1 ENOENT (Нет такого файла или каталога)
[pid 15626] openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 4
[pid 15626] newfstatat(4, "", {st_mode=S_IFREG|0644, st_size=82523, ...}, AT_EMPTY_PATH) =
0
[pid 15626] mmap(NULL, 82523, PROT_READ, MAP_PRIVATE, 4, 0) = 0x7f6b2752e000
[pid 15626] close(4)                    = 0
[pid 15626] openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 4
[pid 15626] read(4, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\0\0\0\0\0"...,
832) = 832
[pid 15626] pread64(4, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"...,
784, 64) = 784
```

```
[pid 15626] pread64(4, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48, 848) = 48
[pid 15626] pread64(4, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0I\17\357\204\3$\f\221\2039x\324\224\323\236S"..., 68, 896) = 68
[pid 15626] newfstatat(4, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, AT_EMPTY_PATH) = 0
[pid 15626] pread64(4, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
[pid 15626] mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 4, 0) = 0x7f6b27305000
[pid 15626] mprotect(0x7f6b2732d000, 2023424, PROT_NONE) = 0
[pid 15626] mmap(0x7f6b2732d000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 4, 0x28000) = 0x7f6b2732d000
[pid 15626] mmap(0x7f6b274c2000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 4, 0x1bd000) = 0x7f6b274c2000
[pid 15626] mmap(0x7f6b2751b000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 4, 0x215000) = 0x7f6b2751b000
[pid 15626] mmap(0x7f6b27521000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f6b27521000
[pid 15626] close(4)                    = 0
[pid 15626] mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f6b27302000
[pid 15626] arch_prctl(ARCH_SET_FS, 0x7f6b27302740) = 0
[pid 15626] set_tid_address(0x7f6b27302a10) = 15626
[pid 15626] set_robust_list(0x7f6b27302a20, 24) = 0
[pid 15626] rseq(0x7f6b273030e0, 0x20, 0, 0x53053053) = 0
[pid 15626] mprotect(0x7f6b2751b000, 16384, PROT_READ) = 0
[pid 15626] mprotect(0x55c27a764000, 4096, PROT_READ) = 0
[pid 15626] mprotect(0x7f6b2757d000, 8192, PROT_READ) = 0
[pid 15626] prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
[pid 15626] munmap(0x7f6b2752e000, 82523) = 0
[pid 15626] read(0, "0", 1)             = 1
[pid 15626] read(0, " ", 1)             = 1
[pid 15626] read(0, "0", 1)             = 1
[pid 15626] read(0, " ", 1)             = 1
[pid 15626] read(0, "0", 1)             = 1
[pid 15626] read(0, " ", 1)             = 1
[pid 15626] read(0, "1", 1)             = 1
[pid 15626] read(0, "\n", 1)            = 1
[pid 15626] write(1, "1", 1)            = 1
[pid 15626] write(1, "\n", 1)           = 1
[pid 15626] read(0, "1", 1)             = 1
[pid 15626] read(0, ".", 1)             = 1
[pid 15626] read(0, "2", 1)             = 1
[pid 15626] read(0, " ", 1)             = 1
[pid 15626] read(0, "1", 1)             = 1
[pid 15626] read(0, "4", 1)             = 1
[pid 15626] read(0, " ", 1)             = 1
[pid 15626] read(0, "1", 1)             = 1
[pid 15626] read(0, "5", 1)             = 1
[pid 15626] read(0, ".", 1)             = 1
[pid 15626] read(0, "6", 1)             = 1
[pid 15626] read(0, "6", 1)             = 1
[pid 15626] read(0, "6", 1)             = 1
[pid 15626] read(0, "\n", 1)            = 1
[pid 15626] write(1, "3", 1)            = 1
```

```
[pid 15626] write(1, "0", 1)           = 1
[pid 15626] write(1, ".", 1)          = 1
[pid 15626] write(1, "8", 1)           = 1
[pid 15626] write(1, "6", 1)           = 1
[pid 15626] write(1, "6", 1)           = 1
[pid 15626] write(1, "\n", 1)          = 1
[pid 15626] read(0, "1", 1)            = 1
[pid 15626] read(0, " ", 1)           = 1
[pid 15626] read(0, "2", 1)            = 1
[pid 15626] read(0, "3", 1)            = 1
[pid 15626] read(0, " ", 1)           = 1
[pid 15626] read(0, "-", 1)           = 1
[pid 15626] read(0, "1", 1)            = 1
[pid 15626] read(0, "6", 1)            = 1
[pid 15626] read(0, "\n", 1)           = 1
[pid 15626] write(1, "8", 1)           = 1
[pid 15626] write(1, "\n", 1)          = 1
[pid 15626] read(0, "-", 1)           = 1
[pid 15626] read(0, "1", 1)            = 1
[pid 15626] read(0, ".", 1)           = 1
[pid 15626] read(0, "4", 1)            = 1
[pid 15626] read(0, " ", 1)           = 1
[pid 15626] read(0, "4", 1)            = 1
[pid 15626] read(0, " ", 1)           = 1
[pid 15626] read(0, "4", 1)            = 1
[pid 15626] read(0, "\n", 1)           = 1
[pid 15626] write(1, "6", 1)           = 1
[pid 15626] write(1, ".", 1)          = 1
[pid 15626] write(1, "6", 1)           = 1
[pid 15626] write(1, "\n", 1)          = 1
[pid 15626] read(0, "", 1)            = 0
[pid 15626] exit_group(0)              = ?
[pid 15626] +++ exited with 0 +++
<... wait4 resumed>NULL, 0, NULL)       = 15626
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=15626, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
read(0, "1", 1)                    = 1
write(1, "1", 11)                    = 1
read(0, "\n", 1)                   = 1
write(1, "\n", 1
)                  = 1
read(0, "3", 1)                    = 1
write(1, "3", 13)                    = 1
read(0, "0", 1)                    = 1
write(1, "0", 10)                    = 1
read(0, ".", 1)                   = 1
write(1, ".", 1.)                  = 1
read(0, "8", 1)                    = 1
write(1, "8", 18)                    = 1
read(0, "6", 1)                    = 1
write(1, "6", 16)                    = 1
read(0, "6", 1)                    = 1
write(1, "6", 16)                    = 1
read(0, "\n", 1)                   = 1
write(1, "\n", 1
)                  = 1
```

```
read(0, "8", 1)                 = 1
write(1, "8", 18)                 = 1
read(0, "\n", 1)                = 1
write(1, "\n", 1
)                 = 1
read(0, "6", 1)                 = 1
write(1, "6", 16)                 = 1
read(0, ".", 1)                = 1
write(1, ".", 1.)               = 1
read(0, "6", 1)                 = 1
write(1, "6", 16)                 = 1
read(0, "\n", 1)                = 1
write(1, "\n", 1
)                 = 1
read(0, "", 1)                  = 0
close(1)                        = 0
close(0)                        = 0
exit_group(0)                     = ?
+++ exited with 0 +++
```

Вывод

В результате обработки файла с командами, состоящими из произвольного количества чисел типа float, дочерний процесс успешно суммирует все указанные значения и выводит полученную сумму в стандартный поток вывода. Это решение позволяет эффективно обрабатывать входные данные, независимо от их объема, благодаря динамическому подходу к чтению и суммированию чисел.