Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №3 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); – создание дочернего процесса
· pid_t wait(int) - ожидание завершения дочерних процессов ·
key_t ftok (const char *, int) – создание ключа System V IPC
· int shmget(key_t, size_t, int) – получение дескриптора (создание) разделяемого сегмента памяти
· void *shmat(int, const void*, int) – внесение разделяемого сегмента памяти в пространство имен процесса
· int shmdt(const void*) - удаленеи разделяемого сегмента памяти из пространства имен процесса
· int shmctl(int, int, struct shmid_ds *) - удаление разделяемого сегмента памяти

Описание работы программы:

1. Родительская программа (main.c):

Инициализация:

· Программа открывает файл, переданный как аргумент командной строки (argv[1]). Если файл не удаётся открыть, выводится сообщение об ошибке.
· Генерируются два ключа с помощью функции ftok() для доступа к разделяемой памяти между родительской и дочерней программами.

Создание дочернего процесса:

· Родительский процесс создаёт дочерний процесс с помощью fork(). ·
Функция fork() возвращает:
· -1 в случае ошибки,
· 0 в дочернем процессе,
· PID дочернего процесса в родительском. ·
Дочерний процесс:

- Он перенаправляет свой стандартный ввод (STDIN_FILENO) на файл, открытый ранее, с помощью dup2(input, STDIN_FILENO).
- Затем выполняется программа child.out (собранная из исходного файла child.c) с помощью execve().

· **Родительский процесс**:

- Родительский процесс ожидает завершения дочернего с помощью waitpid().

Работа с разделяемой памятью:

· После завершения дочернего процесса родитель получает доступ к сегментам разделяемой памяти:

1. output_size_key — сегмент, содержащий количество строк, которые вывел дочерний процесс.
2. output_key — сегмент, содержащий фактические данные (результаты обработки дочерней программы).

· Используется shmget() для получения ID сегментов разделяемой памяти и shmat() для их подключения к пространству адресов процесса.

· Родительская программа затем читает данные из разделяемой памяти и выводит их на стандартный вывод (STDOUT_FILENO).

· В конце программа отсоединяется от разделяемой памяти и закрывает файл.

2. Дочерняя программа (child.c):

Чтение и обработка входных данных:

· Дочерний процесс генерирует ключи для разделяемой памяти (output_size_key и output_key).

· Создаёт два сегмента разделяемой памяти с помощью shmget(): ·

Один для хранения размера вывода (output_size).

· Другой для хранения самих данных вывода.

· Создаётся буфер (output_buff), и дочерний процесс начинает читать данные из STDIN_FILENO (который был перенаправлен на файл родительским процессом).

Обработка чисел:

· Программа читает данные слово за словом с помощью функции read_until_space(), которая считывает символы из файла до первого пробела, табуляции или новой строки.

· Каждое слово пытается преобразовать в **число с плавающей точкой** с помощью функции read_double(). Если слово не является числом, выводится ошибка.

· Числа суммируются в переменной sum. Когда встречается символ новой строки ('\ n'), программа записывает текущую сумму (преобразованную в строку) в разделяемую память и сбрасывает сумму в ноль.

· Этот процесс продолжается до тех пор, пока все входные данные не будут обработаны.

Запись результатов в разделяемую память:

- После обработки всех данных программа записывает результат в разделяемую память в сегмент output_key.
- Программа обновляет переменную *output_size, которая отслеживает количество строк результатов, и увеличивает размер выделенного буфера, если это необходимо.
- В конце дочерний процесс отсоединяется от разделяемой памяти (shmdt()), освобождает динамически выделенную память (free(output_buff)) и закрывает стандартный ввод.

Основные функции:

- **Работа с разделяемой памятью:**
  - shmem_create() — создаёт или получает доступ к сегменту разделяемой памяти для записи данных.
  - shmem_get() — получает ID существующего сегмента разделяемой памяти для чтения.
  - shmat() и shmdt() — подключают и отсоединяют разделяемую память от процесса.
- **Обработка ошибок:**
  - write_error() — выводит сообщения об ошибке в стандартный поток ошибок (STDERR_FILENO).
- **Обработка данных:**
  - read_double() — преобразует строку в число с плавающей точкой, обрабатывая как целую, так и дробную части.
  - read_until_space() — считывает символы из стандартного ввода до пробела или новой строки и записывает их в буфер.

Поток данных:

1. **Родительский процесс**:
   - Открывает входной файл.
   - Создаёт дочерний процесс и ожидает его завершения.
   - Читает результаты из разделяемой памяти и выводит их на экран.
2. **Дочерний процесс**:
   - Читает данные из файла.
   - Преобразует каждое число в формат double, суммирует их и записывает результаты в разделяемую память.

Обработка ошибок:

Обе программы включают обработку ошибок для распространённых проблем, таких как:

- Ошибки открытия файлов.
- Ошибки выделения разделяемой памяти.
- Ошибки при вызовах системных функций (fork(), execve() и т. д.).

Общий рабочий процесс:

1. Родительская программа подготавливает и открывает файл.
2. Родитель создаёт дочерний процесс, который читает числа, суммирует их и записывает результаты в разделяемую память.
3. После завершения дочернего процесса родитель забирает результаты из разделяемой памяти и выводит их.

Код программы

main.c:

```
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>

#include <sys/shm.h>
#include <sys/ipc.h>
#include <errno.h>



void write_error(const char *msg) {
    write(STDERR_FILENO, msg, sizeof(msg) - 1);
}

int generate_keys(int *key1, int *key2) {
    if ((*key1 = ftok("main.c", 0)) == -1) {
        write_error("ftok error\n");
        return -1;
    }
    if ((*key2 = ftok("child.c", 0)) == -1) {
        write_error("ftok error\n");
```

```
        return -1;
    }
    return 0;
}

int shmem_get(int *shmid, int key, size_t size) {
    if ((*shmid = shmget(key, size, SHM_RDONLY)) == -1) {
        write_error("shmget error\n");
        return -1;
    }
    return 0;
}



int main(int argc, char* argv[]) {
    int input;
    input = open(argv[1], O_RDONLY);
    if (input < 0) {
        write_error("file error\n");
        return -1;
    }

    int output_size_key;
    int output_key;
    if (generate_keys(&output_size_key, &output_key)) {
        return -1;
    }

    pid_t child;
    child = fork();

    switch (child)
```

```
{
case -1:
    write_error("fork error\n");
    return -1;

case 0:
    dup2(input, STDIN_FILENO);

    if (execve("child.out", argv, NULL) != 0) {
        write_error("execve error\n");
        return -1;
    }
    break;

default:
    waitpid(child, NULL, 0);
}

int output_size_shmid;
if (shmem_get(&output_size_shmid, output_size_key, sizeof(size_t))) {
    return -1;
}
size_t *output_size;
if ((output_size = (size_t *)shmat(output_size_shmid, NULL, 0)) == NULL) {
    write_error("shmat error\n");
    return -1;
}

int output_shmid;
if (shmem_get(&output_shmid, output_key, *output_size * sizeof(char))) {
    return -1;
}
```

```
    char *output;
    if ((output = (char *)shmat(output_shmid, NULL, 0)) == NULL) {
        write_error("shmat error\n");
        return -1;
    }

    write(STDOUT_FILENO, output, (strlen(output)) * sizeof(char));

    if (shmctl(output_size_shmid, 0, NULL) == -1) {
        write_error("shmctl error\n");
        return -1;
    }

    if (shmctl(output_shmid, 0, NULL) == -1) {
        write_error("shmctl error\n");
        return -1;
    }

    close(input);

    return 0;
}
```

child.c:

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

#include <sys/shm.h>
```

```
#include <sys/ipc.h>
#include <errno.h>

#include <stdio.h>

void write_error(const char *msg) {
    write(STDERR_FILENO, msg, sizeof(msg) - 1);
}

int read_double(char* inp, double* number) {
    double int_part = 0;
    double frac_part = 0;
    double sign = 1;
    if (*inp == '-') {
        sign = -1;
        ++inp;
    }

    while (*inp != '.' && *inp != ',' && *inp != 0) {
        if (*inp < '0' || *inp > '9') {

            return 3;
        }
        int_part *= 10;
        int_part += (*inp - '0');
        ++inp;
    }
    if (*inp == 0) {
        *number = sign * int_part;
        return 0;
    }
    while (*inp != 0) {++inp;}
```

```
    --inp;
    while (*inp != '.' && *inp != ',') {
        if (*inp < '0' || *inp > '9' || *inp == 0) {
            return 3;
        }
        frac_part += (*inp - '0');
        frac_part /= 10;
        --inp;
    }
    *number = sign * (int_part + frac_part);
    return 0;
}

int read_until_space(int fd, char *targ, char *c) {
    char *ptr = targ;
    while (1) {
        if (read(fd, c, sizeof(char)) == 0) {
            *ptr = 0;
            *c = '\n';
            return 1;
        }
        if (*c == ' ' || *c == '\t' || *c == '\n') {
            *ptr = 0;
            return 0;
        }
        *ptr = *c;
        ++ptr;
    }
}

int generate_keys(int *key1, int *key2) {
    if ((*key1 = ftok("main.c", 0)) == -1) {
```

```
        write_error("ftok error\n");
        return -1;
    }
    if ((*key2 = ftok("child.c", 0)) == -1) {
        write_error("ftok error\n");
        return -1;
    }
    return 0;
}

int shmem_create(int *shmid, int key, size_t size) {
    if ((*shmid = shmget(key, size, IPC_CREAT|IPC_EXCL|0666)) == -1) {
        if (errno != EEXIST) {
            write_error("shmget error\n");
            return -1;
        }

        if ((*shmid = shmget(key, size, 0666)) == -1) {
            write_error("shmget error\n");
            return -1;
        }
        if (shmctl(*shmid, 0, NULL) < 0) {
            write_error("shmctl error\n");
            return -1;
        }
        if ((*shmid = shmget(key, size, IPC_CREAT|IPC_EXCL|0666)) == -1) {
            write_error("shmget error\n");
            return -1;
        }
    }
    return 0;
}
```

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

    int output_size_key;
    int output_key;
    if (generate_keys(&output_size_key, &output_key)) {
        return -1;
    }

    int output_size_shmid;
    if (shmem_create(&output_size_shmid, output_size_key, sizeof(size_t))) {
        return -1;
    }
    size_t *output_size;
    if ((output_size = (size_t *)shmat(output_size_shmid, NULL, 0)) == NULL) {
        write_error("shmat error\n");
        return -1;
    }
    *output_size = 0;

    char c = ' ';

    char word[64];
    char *ptr = word;

    char *output_buff;
    size_t len = 256;
    if ((output_buff = (char *)malloc(len * sizeof(char))) == NULL) {
        write_error("malloc error\n");
        return -1;
    }
```

```
double num;
double sum = 0;
int f = 1;
char *w = output_buff;

while (f) {
    if (read_until_space(STDIN_FILENO, word, &c)) {
        f = 0;
    }
    if (strlen(word) == 0) {
        break;
    }
    if ((read_double(word, &num)) != 0) {
        write_error("error reading double");
        return -1;
    }

    sum += num;
    ptr = word;

    if (c == '\n') {

        sprintf(word, "%g", sum);
        sum = 0;

        strcpy(w, word);

        w[strlen(w)] = '\n';
        w[strlen(w)] = 0;
        w = &(w[strlen(w)]);
```

```
        (*output_size)++;

        if (len - strlen(output_buff) < 64) {
            len *= 2;
            output_buff = (char *)realloc(output_buff, len * sizeof(char));
            if (output_buff == NULL) {
                write_error("realloc error\n");
                return -1;
            }
        }
    }
}
int output_shmid;
if (shmem_create(&output_shmid, output_key, *output_size * sizeof(char))) {
    return -1;
}
char *output;
if ((output = (char *)shmat(output_shmid, NULL, 0)) == NULL) {
    write_error("shmgat error\n");
    return -1;
}

strcpy(output, output_buff);

if (shmdt(output) != 0) {
    write_error("shmdt error\n");
    return -1;
}
if (shmdt(output_size) != 0) {
    write_error("shmdt error\n");
    return -1;
}
```

```
    free(output_buff);
    close(STDIN_FILENO);


    return 0;
}
```

Протокол работы программы

Вывод утилиты strace:

```
execve("./solution.out", ["./solution.out", "./inp.txt"], 0x7fff69934020 /* 68 vars */) = 0

brk(NULL)                               = 0x5afe2989f000

arch_prctl(0x3001 /* ARCH_??? */, 0x7ffc275717e0) = -1 EINVAL (Invalid argument)

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7725a37c1000

access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=77915, ...}, AT_EMPTY_PATH) = 0

mmap(NULL, 77915, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7725a37ad000

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) = 0x7725a3400000

mprotect(0x7725a3428000, 2023424, PROT_NONE) = 0
```

mmap(0x7725a3428000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7725a3428000

mmap(0x7725a35bd000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) = 0x7725a35bd000

mmap(0x7725a3616000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x7725a3616000

mmap(0x7725a361c000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7725a361c000

close(3)   = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7725a37aa000

arch_prctl(ARCH_SET_FS, 0x7725a37aa740) = 0

set_tid_address(0x7725a37aaa10)   = 6325

set_robust_list(0x7725a37aaa20, 24)   = 0

rseq(0x7725a37ab0e0, 0x20, 0, 0x53053053) = 0

mprotect(0x7725a3616000, 16384, PROT_READ) = 0

mprotect(0x5afe2935e000, 4096, PROT_READ) = 0

mprotect(0x7725a37fb000, 8192, PROT_READ) = 0

prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0

munmap(0x7725a37ad000, 77915)   = 0

openat(AT_FDCWD, "./inp.txt", O_RDONLY) = 3

newfstatat(AT_FDCWD, "main.c", {st_mode=S_IFREG|0664, st_size=2322, ...}, 0) = 0

newfstatat(AT_FDCWD, "child.c", {st_mode=S_IFREG|0664, st_size=4761, ...}, 0) = 0

clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLDstrace: Process 6326 attached

, child_tidptr=0x7725a37aaa10) = 6326

[pid 6325] wait4(6326, <unfinished ...>

[pid 6326] set_robust_list(0x7725a37aaa20, 24) = 0

[pid 6326] dup2(3, 0)   = 0

[pid 6326] execve("child.out", ["./solution.out", "./inp.txt"], NULL) = 0

[pid 6326] brk(NULL)   = 0x5eff88014000

[pid 6326] arch_prctl(0x3001 /* ARCH_??? */, 0x7ffd8a82c460) = -1 EINVAL (Invalid argument)

[pid 6326] mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x71c8fe075000

[pid 6326] access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)

[pid 6326] openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 4

[pid 6326] newfstatat(4, "", {st_mode=S_IFREG|0644, st_size=77915, ...}, AT_EMPTY_PATH) = 0

[pid 6326] mmap(NULL, 77915, PROT_READ, MAP_PRIVATE, 4, 0) = 0x71c8fe061000

[pid 6326] close(4)   = 0

[pid 6326] openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 4

[pid 6326] 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 6326] 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 6326] 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 6326] 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 6326] newfstatat(4, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, AT_EMPTY_PATH) = 0

[pid 6326] 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 6326] mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 4, 0) = 0x71c8fde00000

[pid 6326] mprotect(0x71c8fde28000, 2023424, PROT_NONE) = 0

[pid 6326] mmap(0x71c8fde28000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 4, 0x28000) = 0x71c8fde28000

[pid 6326] mmap(0x71c8fdfbd000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 4, 0x1bd000) = 0x71c8fdfbd000

[pid 6326] mmap(0x71c8fe016000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 4, 0x215000) = 0x71c8fe016000

[pid 6326] mmap(0x71c8fe01c000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x71c8fe01c000

[pid 6326] close(4) = 0

[pid 6326] mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x71c8fe05e000

[pid 6326] arch_prctl(ARCH_SET_FS, 0x71c8fe05e740) = 0

[pid 6326] set_tid_address(0x71c8fe05ea10) = 6326

[pid 6326] set_robust_list(0x71c8fe05ea20, 24) = 0

[pid 6326] rseq(0x71c8fe05f0e0, 0x20, 0, 0x53053053) = 0

[pid 6326] mprotect(0x71c8fe016000, 16384, PROT_READ) = 0

[pid 6326] mprotect(0x5eff872b8000, 4096, PROT_READ) = 0

[pid 6326] mprotect(0x71c8fe0af000, 8192, PROT_READ) = 0

[pid 6326] prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0

[pid 6326] munmap(0x71c8fe061000, 77915) = 0

[pid 6326] newfstatat(AT_FDCWD, "main.c", {st_mode=S_IFREG|0664, st_size=2322, ...}, 0) = 0

[pid 6326] newfstatat(AT_FDCWD, "child.c", {st_mode=S_IFREG|0664, st_size=4761, ...}, 0) = 0

[pid 6326] shmget(0x53a6c, 8, IPC_CREAT|IPC_EXCL|0666) = 44

[pid 6326] shmat(44, NULL, 0) = 0x71c8fe0ae000

[pid 6326] getrandom("\xf7\x9a\x03\xa4\x23\x6d\x60\x6b", 8, GRND_NONBLOCK) = 8

[pid 6326] brk(NULL) = 0x5eff88014000

[pid 6326] brk(0x5eff88035000) = 0x5eff88035000

[pid 6326] read(0, "0", 1) = 1

[pid 6326] read(0, " ", 1) = 1

```
[pid 6326] read(0, "0", 1)              = 1
[pid 6326] read(0, " ", 1)              = 1
[pid 6326] read(0, "0", 1)              = 1
[pid 6326] read(0, " ", 1)              = 1
[pid 6326] read(0, "1", 1)              = 1
[pid 6326] read(0, "\n", 1)             = 1
[pid 6326] read(0, "1", 1)              = 1
[pid 6326] read(0, ".", 1)              = 1
[pid 6326] read(0, "2", 1)              = 1
[pid 6326] read(0, " ", 1)              = 1
[pid 6326] read(0, "1", 1)              = 1
[pid 6326] read(0, "4", 1)              = 1
[pid 6326] read(0, " ", 1)              = 1
[pid 6326] read(0, "1", 1)              = 1
[pid 6326] read(0, "5", 1)              = 1
[pid 6326] read(0, ".", 1)              = 1
[pid 6326] read(0, "6", 1)              = 1
[pid 6326] read(0, "6", 1)              = 1
[pid 6326] read(0, "6", 1)              = 1
[pid 6326] read(0, "\n", 1)             = 1
[pid 6326] read(0, "1", 1)              = 1
[pid 6326] read(0, " ", 1)              = 1
[pid 6326] read(0, "2", 1)              = 1
[pid 6326] read(0, "3", 1)              = 1
[pid 6326] read(0, " ", 1)              = 1
[pid 6326] read(0, "-", 1)              = 1
[pid 6326] read(0, "1", 1)              = 1
```

[pid 6326] read(0, "6", 1) = 1

[pid 6326] read(0, "\n", 1) = 1

[pid 6326] read(0, "-", 1) = 1

[pid 6326] read(0, "1", 1) = 1

[pid 6326] read(0, ".", 1) = 1

[pid 6326] read(0, "4", 1) = 1

[pid 6326] read(0, " ", 1) = 1

[pid 6326] read(0, "4", 1) = 1

[pid 6326] read(0, " ", 1) = 1

[pid 6326] read(0, "4", 1) = 1

[pid 6326] read(0, "", 1) = 0

[pid 6326] shmget(0x5516a, 4, IPC_CREAT|IPC_EXCL|0666) = 45

[pid 6326] shmat(45, NULL, 0) = 0x71c8fe074000

[pid 6326] shmdt(0x71c8fe074000) = 0

[pid 6326] shmdt(0x71c8fe0ae000) = 0

[pid 6326] close(0) = 0

[pid 6326] exit_group(0) = ?

[pid 6326] +++ exited with 0 +++

<... wait4 resumed>NULL, 0, NULL) = 6326

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=6326, si_uid=1000, si_status=0, si_utime=0, si_stime=0} ---

shmget(0x53a6c, 8, SHM_NORESERVE|000) = 44

shmat(44, NULL, 0) = 0x7725a37fa000

shmget(0x5516a, 4, SHM_NORESERVE|000) = 45

shmat(45, NULL, 0) = 0x7725a37c0000

write(1, "1\n30.866\n8\n6.6\n", 151

30.866

8

6.6

) = 15

shmctl(44, IPC_RMID, NULL) = 0

shmctl(45, IPC_RMID, NULL) = 0

close(3) = 0

exit_group(0) = ?

+++ exited with 0 +++

Вывод

Написал и отладил программу на языке си, реализующую алгоритм в соответсвии с заданием варианта, с использованием разделяемой памяти.