Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

**Лабораторная работа №2 по курсу**

**«Операционные системы»**

Группа: М8О-215Б-23

Студент: Верменников М.В.

Преподаватель: Миронов Е.С.

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

Дата: 20.09.24

Москва, 2023

**Постановка задачи**

**Вариант 4.**

Пользователь вводит команды вида: «число число число<endline>». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс производит деление первого числа, на последующие, а результат выводит в файл. Если происходит деление на 0, то тогда дочерний и родительский процесс завершают свою работу. Проверка деления на 0 должна осуществляться на стороне дочернего процесса. 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); – создаёт канал (pipe) для межпроцессного взаимодействия.
* int dup2(int oldfd, int newfd); - дублирует файловый дескриптор oldfd и заменяет им дескриптор newfd.
* int execl(const char \*path, const char \*arg, ...); - загружает и исполняет новый образ программы.
* int waitpid(pid\_t pid, int \*status, int options); - ожидает завершения дочернего процесса с идентификатором pid и получает его статус завершения.
* ssize\_t read(int fd, void \*buf, size\_t count); - читает данные из файлового дескриптора fd и сохраняет их в буфер buf.
* ssize\_t write(int fd, const void \*buf, size\_t count); - записывает данные из буфера buf в файловый дескриптор fd.

### Алгоритм работы программы:

1. **Создание pipe (каналов связи):**

В начале программы создаются два канала с помощью системного вызова pipe(). Первый канал (fd1) используется для передачи данных от родительского процесса к дочернему, второй канал (fd2) — для передачи результатов от дочернего процесса обратно родительскому.

1. **Создание дочернего процесса:**

Далее с помощью системного вызова fork() создается дочерний процесс. На этом этапе система разделяет процесс на два: родительский и дочерний. В результате вызова:

* + - Родительский процесс получает PID дочернего процесса.
    - Дочерний процесс получает значение 0.

1. **Родительский процесс:**

После вызова fork(), родительский процесс:

* + - Закрывает ненужные дескрипторы: конец чтения первого pipe и конец записи второго pipe.
    - Ожидает ввода чисел от пользователя, который вводит их через стандартный ввод. Эти числа собираются в вектор и затем передаются дочернему процессу через первый pipe.
    - После отправки данных родительский процесс ожидает завершения работы дочернего с помощью системного вызова waitpid().
    - По завершении работы дочернего процесса родительский процесс читает результаты из второго pipe (который дочерний процесс использовал для отправки результатов).
    - Выводит полученные результаты на экран.

1. **Дочерний процесс:**

Дочерний процесс после вызова fork():

* + - Закрывает ненужные дескрипторы: конец записи первого pipe и конец чтения второго pipe.
    - Перенаправляет стандартный ввод и вывод на соответствующие концы pipe с помощью системного вызова dup2(), так что данные будут передаваться через pipe.
    - Загружает новую программу с помощью execl(), заменяя свой код программой, которая будет читать данные из первого pipe и записывать результаты во второй pipe.
    - В программе child происходит обработка переданных данных: выполняется деление первого числа на последующие, а результат записывается в файл и передается родительскому процессу через второй pipe.
    - В случае деления на 0, программа завершает работу с соответствующим кодом ошибки.

1. **Межпроцессное взаимодействие:**

Передача данных между процессами осуществляется через pipe:

* + - Родительский процесс пишет данные в pipe1, которые дочерний процесс считывает.
    - Дочерний процесс обрабатывает данные и передает результаты обратно через pipe2.
    - Родительский процесс получает результаты и выводит их.

1. **Завершение работы:**

После завершения работы дочернего процесса, родительский процесс получает результаты и выводит их на экран. Программа завершается.

**Код программы**

**Main.cpp**

#include <iostream>

#include <vector>

#include <unistd.h>

#include <sys/wait.h>

#include <fcntl.h>

int main() {

    int fd1[2], fd2[2];

    if (pipe(fd1) == -1 || pipe(fd2) == -1) {

        std::cout << "Ошибка при создании pipe";

        return 1;

    }

    pid\_t pid = fork();

    if (pid < 0) {

        std::cout << "Ошибка при создании процесса";

        return 1;

    }

    if (pid > 0) { // Родительский процесс

        close(fd1[0]); // Закрываем чтение из первого pipe

        close(fd2[1]); // Закрываем запись во второй pipe

        std::cout << "Введите числа в формате: «число число число<endline>»." << std::endl;

        std::vector<float> numbers;

        float number;

        while (std::cin >> number) {

            numbers.push\_back(number);

            if (std::cin.peek() == '\n') break; // Сравнивается с помощью cin.peek() последующий символ, не извлекаясь. То есть если пользователь нажал enter, то ввод завершён

        }

        // Передача чисел дочернему процессу

        int len = numbers.size();

        if (write(fd1[1], &len, sizeof(int)) == -1) {

            std::cout << "Ошибка при записи длины";

            return 1;

        }

        if (write(fd1[1], numbers.data(), len \* sizeof(float)) == -1) {

            std::cout << "Ошибка при записи чисел";

            return 1;

        }

        // Ожидание завершения дочернего процесса

        int status;

        waitpid(pid, &status, 0);

        if (WIFEXITED(status) && WEXITSTATUS(status) == 1) {

            std::cerr << "Деление на 0. Завершение работы." << std::endl;

            return 1;

        }

        // Чтение результата

        int res\_len = len - 1;

        std::vector<float> results(res\_len);

        if (read(fd2[0], results.data(), res\_len \* sizeof(float)) == -1) {

            std::cout << "Ошибка при чтении результатов";

            return 1;

        }

        std::cout << "Результат деления: ";

        std::cout << results.back() << " ";

        std::cout << std::endl;

        close(fd1[1]);

        close(fd2[0]);

    } else { // Дочерний процесс

        close(fd1[1]); // Закрываем запись в первый pipe

        close(fd2[0]); // Закрываем чтение из второго pipe

        // Перенаправление stdin и stdout на pipe

        if (dup2(fd1[0], STDIN\_FILENO) == -1) {

            std::cout << "Ошибка при перенаправлении stdin";

            return 1;

        }

        if (dup2(fd2[1], STDOUT\_FILENO) == -1) {

            std::cout << "Ошибка при перенаправлении stdout";

            return 1;

        }

        // Запуск дочернего процесса

        execl("./child", "child", NULL);

        std::cout << "Ошибка при вызове execl";

        return 1;

    }

    return 0;

}

**Child.cpp**

#include <unistd.h>

#include <vector>

#include <fcntl.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <iostream>

#include <cstdio>

int main() {

    // Чтение чисел из stdin

    int len;

    if (read(STDIN\_FILENO, &len, sizeof(int)) == -1) {

        std::cout <<  "Ошибка при чтении длины";

        return 1;

    }

    std::vector<float> numbers(len);

    if (read(STDIN\_FILENO, numbers.data(), len \* sizeof(float)) == -1) {

        std::cout <<  "Ошибка при чтении чисел";

        return 1;

    }

    // Открытие файла для записи

    int fd = open("result.txt", O\_WRONLY | O\_CREAT | O\_TRUNC, 0666);

    if (fd == -1) {

        std::cout << "Ошибка при открытии файла";

        return 1;

    }

    // Выполнение деления и запись результатов

    float first\_number = numbers[0];

    int res\_len = len - 1;

    std::vector<float> results(res\_len);

    for (int i = 1; i < len; ++i) {

        if (numbers[i] == 0) {

            close(fd);

            exit(1);  // Деление на 0

        }

        results[i - 1] = first\_number / numbers[i];

        if (i == len - 1) {

            // Форматирование результата в строку

            char buffer[64];

            int n = snprintf(buffer, sizeof(buffer), "%f\n", results[i - 1]);

            if (n <= 0) {

                std::cout << "Ошибка при форматировании строки";

                close(fd);

                return 1;

            }

            // Запись строки в файл

            if (write(fd, buffer, n) == -1) {

                std::cout << "Ошибка при записи в файл";

                close(fd);

                return 1;

            }

        }

    }

    close(fd);

    // Передача результатов родительскому процессу через stdout

    if (write(STDOUT\_FILENO, results.data(), res\_len \* sizeof(float)) == -1) {

        std::cout << "Ошибка при записи результатов";

        return 1;

    }

    return 0;

}

**Протокол работы программы**

**Тестирование:**

$ ./main

Введите числа в формате: «число число число<endline>».

4 4 5 5 6

Результат деления: 0.666667

$ cat < result.txt

0.666667

**Strace:**

**13589 execve("./main", ["./main"], 0x7ffe92e59918 /\* 26 vars \*/) = 0**

13589 brk(NULL) = 0x9d5000

13589 mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7fbef63e9000

13589 access("/etc/ld.so.preload", R\_OK) = -1 ENOENT (No such file or directory)

13589 openat(AT\_FDCWD, "/etc/ld.so.cache", O\_RDONLY|O\_CLOEXEC) = 3

13589 newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=25258, ...}, AT\_EMPTY\_PATH) = 0

13589 mmap(NULL, 25258, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7fbef63e2000

**13589 close(3) = 0**

13589 openat(AT\_FDCWD, "/usr/local/lib64/libstdc++.so.6", O\_RDONLY|O\_CLOEXEC) = 3

13589 read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832

13589 newfstatat(3, "", {st\_mode=S\_IFREG|0755, st\_size=2530008, ...}, AT\_EMPTY\_PATH) = 0

13589 mmap(NULL, 2543808, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7fbef6174000

13589 mmap(0x7fbef6219000, 1216512, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0xa5000) = 0x7fbef6219000

13589 mmap(0x7fbef6342000, 581632, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1ce000) = 0x7fbef6342000

13589 mmap(0x7fbef63d0000, 57344, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x25c000) = 0x7fbef63d0000

13589 mmap(0x7fbef63de000, 12480, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7fbef63de000

**13589 close(3) = 0**

13589 openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libm.so.6", O\_RDONLY|O\_CLOEXEC) = 3

13589 read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832

13589 newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=907784, ...}, AT\_EMPTY\_PATH) = 0

13589 mmap(NULL, 909560, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7fbef6095000

13589 mmap(0x7fbef60a5000, 471040, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x10000) = 0x7fbef60a5000

13589 mmap(0x7fbef6118000, 368640, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x83000) = 0x7fbef6118000

13589 mmap(0x7fbef6172000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0xdc000) = 0x7fbef6172000

**13589 close(3) = 0**

13589 openat(AT\_FDCWD, "/usr/local/lib64/libgcc\_s.so.1", O\_RDONLY|O\_CLOEXEC) = 3

13589 read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832

13589 newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=906528, ...}, AT\_EMPTY\_PATH) = 0

13589 mmap(NULL, 181160, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7fbef6068000

13589 mmap(0x7fbef606c000, 143360, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x4000) = 0x7fbef606c000

13589 mmap(0x7fbef608f000, 16384, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x27000) = 0x7fbef608f000

13589 mmap(0x7fbef6093000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x2b000) = 0x7fbef6093000

**13589 close(3) = 0**

13589 openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libc.so.6", O\_RDONLY|O\_CLOEXEC) = 3

13589 read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\20t\2\0\0\0\0\0"..., 832) = 832

13589 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

13589 newfstatat(3, "", {st\_mode=S\_IFREG|0755, st\_size=1922136, ...}, AT\_EMPTY\_PATH) = 0

13589 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

13589 mmap(NULL, 1970000, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7fbef5e87000

13589 mmap(0x7fbef5ead000, 1396736, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x26000) = 0x7fbef5ead000

13589 mmap(0x7fbef6002000, 339968, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x17b000) = 0x7fbef6002000

13589 mmap(0x7fbef6055000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1ce000) = 0x7fbef6055000

13589 mmap(0x7fbef605b000, 53072, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7fbef605b000

**13589 close(3) = 0**

13589 mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7fbef5e85000

13589 arch\_prctl(ARCH\_SET\_FS, 0x7fbef5e86480) = 0

13589 set\_tid\_address(0x7fbef5e86750) = 13589

13589 set\_robust\_list(0x7fbef5e86760, 24) = 0

13589 rseq(0x7fbef5e86da0, 0x20, 0, 0x53053053) = 0

13589 mprotect(0x7fbef6055000, 16384, PROT\_READ) = 0

13589 mprotect(0x7fbef6093000, 4096, PROT\_READ) = 0

13589 mprotect(0x7fbef6172000, 4096, PROT\_READ) = 0

13589 mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7fbef5e83000

13589 mprotect(0x7fbef63d0000, 45056, PROT\_READ) = 0

13589 mprotect(0x404000, 4096, PROT\_READ) = 0

13589 mprotect(0x7fbef641b000, 8192, PROT\_READ) = 0

13589 prlimit64(0, RLIMIT\_STACK, NULL, {rlim\_cur=8192\*1024, rlim\_max=RLIM64\_INFINITY}) = 0

13589 munmap(0x7fbef63e2000, 25258) = 0

13589 futex(0x7fbef63de73c, FUTEX\_WAKE\_PRIVATE, 2147483647) = 0

13589 getrandom("\xef\xdf\x83\xf7\x6a\x93\xa5\x74", 8, GRND\_NONBLOCK) = 8

13589 brk(NULL) = 0x9d5000

13589 brk(0x9f6000) = 0x9f6000

13589 pipe2([3, 4], 0) = 0

13589 pipe2([5, 6], 0) = 0

13589 clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7fbef5e86750) = 13590

13590 set\_robust\_list(0x7fbef5e86760, 24 <unfinished ...>

**13589 close(3 <unfinished ...>**

13590 <... set\_robust\_list resumed>) = 0

**13589 <... close resumed>) = 0**

**13590 close(4 <unfinished ...>**

**13589 close(6 <unfinished ...>**

**13590 <... close resumed>) = 0**

**13589 <... close resumed>) = 0**

**13590 close(5 <unfinished ...>**

**13589 newfstatat(1, "", <unfinished ...>**

**13590 <... close resumed>) = 0**

13589 <... newfstatat resumed>{st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0), ...}, AT\_EMPTY\_PATH) = 0

**13590 dup2(3, 0 <unfinished ...>**

13589 write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \321\207\320\270\321\201\320\273\320\260 \320\262 \321\204\320"..., 92 <unfinished ...>

**13590 <... dup2 resumed>) = 0**

13589 <... write resumed>) = 92

**13590 dup2(6, 1 <unfinished ...>**

13589 newfstatat(0, "", <unfinished ...>

**13590 <... dup2 resumed>) = 1**

13589 <... newfstatat resumed>{st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0), ...}, AT\_EMPTY\_PATH) = 0

**13590 execve("./child", ["child"], 0x7ffce21142d8 /\* 26 vars \*/ <unfinished ...>**

13589 read(0, <unfinished ...>

**13590 <... execve resumed>) = 0**

13590 brk(NULL) = 0x19a7000

13590 mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f95e6f3b000

13590 access("/etc/ld.so.preload", R\_OK) = -1 ENOENT (No such file or directory)

13590 openat(AT\_FDCWD, "/etc/ld.so.cache", O\_RDONLY|O\_CLOEXEC) = 4

13590 newfstatat(4, "", {st\_mode=S\_IFREG|0644, st\_size=25258, ...}, AT\_EMPTY\_PATH) = 0

13590 mmap(NULL, 25258, PROT\_READ, MAP\_PRIVATE, 4, 0) = 0x7f95e6f34000

**13590 close(4) = 0**

13590 openat(AT\_FDCWD, "/usr/local/lib64/libstdc++.so.6", O\_RDONLY|O\_CLOEXEC) = 4

13590 read(4, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832

13590 newfstatat(4, "", {st\_mode=S\_IFREG|0755, st\_size=2530008, ...}, AT\_EMPTY\_PATH) = 0

13590 mmap(NULL, 2543808, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 4, 0) = 0x7f95e6cc6000

13590 mmap(0x7f95e6d6b000, 1216512, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 4, 0xa5000) = 0x7f95e6d6b000

13590 mmap(0x7f95e6e94000, 581632, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 4, 0x1ce000) = 0x7f95e6e94000

13590 mmap(0x7f95e6f22000, 57344, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 4, 0x25c000) = 0x7f95e6f22000

13590 mmap(0x7f95e6f30000, 12480, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f95e6f30000

**13590 close(4) = 0**

13590 openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libm.so.6", O\_RDONLY|O\_CLOEXEC) = 4

13590 read(4, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832

13590 newfstatat(4, "", {st\_mode=S\_IFREG|0644, st\_size=907784, ...}, AT\_EMPTY\_PATH) = 0

13590 mmap(NULL, 909560, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 4, 0) = 0x7f95e6be7000

13590 mmap(0x7f95e6bf7000, 471040, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 4, 0x10000) = 0x7f95e6bf7000

13590 mmap(0x7f95e6c6a000, 368640, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 4, 0x83000) = 0x7f95e6c6a000

13590 mmap(0x7f95e6cc4000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 4, 0xdc000) = 0x7f95e6cc4000

**13590 close(4) = 0**

13590 openat(AT\_FDCWD, "/usr/local/lib64/libgcc\_s.so.1", O\_RDONLY|O\_CLOEXEC) = 4

13590 read(4, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832

13590 newfstatat(4, "", {st\_mode=S\_IFREG|0644, st\_size=906528, ...}, AT\_EMPTY\_PATH) = 0

13590 mmap(NULL, 181160, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 4, 0) = 0x7f95e6bba000

13590 mmap(0x7f95e6bbe000, 143360, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 4, 0x4000) = 0x7f95e6bbe000

13590 mmap(0x7f95e6be1000, 16384, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 4, 0x27000) = 0x7f95e6be1000

13590 mmap(0x7f95e6be5000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 4, 0x2b000) = 0x7f95e6be5000

**13590 close(4) = 0**

13590 openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libc.so.6", O\_RDONLY|O\_CLOEXEC) = 4

13590 read(4, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\20t\2\0\0\0\0\0"..., 832) = 832

13590 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

13590 newfstatat(4, "", {st\_mode=S\_IFREG|0755, st\_size=1922136, ...}, AT\_EMPTY\_PATH) = 0

13590 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

13590 mmap(NULL, 1970000, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 4, 0) = 0x7f95e69d9000

13590 mmap(0x7f95e69ff000, 1396736, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 4, 0x26000) = 0x7f95e69ff000

13590 mmap(0x7f95e6b54000, 339968, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 4, 0x17b000) = 0x7f95e6b54000

13590 mmap(0x7f95e6ba7000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 4, 0x1ce000) = 0x7f95e6ba7000

13590 mmap(0x7f95e6bad000, 53072, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f95e6bad000

**13590 close(4) = 0**

13590 mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f95e69d7000

13590 arch\_prctl(ARCH\_SET\_FS, 0x7f95e69d8480) = 0

13590 set\_tid\_address(0x7f95e69d8750) = 13590

13590 set\_robust\_list(0x7f95e69d8760, 24) = 0

13590 rseq(0x7f95e69d8da0, 0x20, 0, 0x53053053) = 0

13590 mprotect(0x7f95e6ba7000, 16384, PROT\_READ) = 0

13590 mprotect(0x7f95e6be5000, 4096, PROT\_READ) = 0

13590 mprotect(0x7f95e6cc4000, 4096, PROT\_READ) = 0

13590 mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f95e69d5000

13590 mprotect(0x7f95e6f22000, 45056, PROT\_READ) = 0

13590 mprotect(0x403000, 4096, PROT\_READ) = 0

13590 mprotect(0x7f95e6f6d000, 8192, PROT\_READ) = 0

13590 prlimit64(0, RLIMIT\_STACK, NULL, {rlim\_cur=8192\*1024, rlim\_max=RLIM64\_INFINITY}) = 0

13590 munmap(0x7f95e6f34000, 25258) = 0

13590 futex(0x7f95e6f3073c, FUTEX\_WAKE\_PRIVATE, 2147483647) = 0

13590 getrandom("\x17\x99\x9f\xb1\xb6\xe7\x63\x8e", 8, GRND\_NONBLOCK) = 8

13590 brk(NULL) = 0x19a7000

13590 brk(0x19c8000) = 0x19c8000

13590 read(0, <unfinished ...>

13589 <... read resumed>"4 4 5 5 6\n", 1024) = 10

13589 write(4, "\5\0\0\0", 4) = 4

13590 <... read resumed>"\5\0\0\0", 4) = 4

13589 write(4, "\0\0\200@\0\0\200@\0\0\240@\0\0\240@\0\0\300@", 20 <unfinished ...>

13590 read(0, <unfinished ...>

13589 <... write resumed>) = 20

13590 <... read resumed>"\0\0\200@\0\0\200@\0\0\240@\0\0\240@\0\0\300@", 20) = 20

13589 wait4(13590, <unfinished ...>

13590 openat(AT\_FDCWD, "result.txt", O\_WRONLY|O\_CREAT|O\_TRUNC, 0666) = 4

13590 write(4, "0.666667\n", 9) = 9

**13590 close(4) = 0**

13590 write(1, "\0\0\200?\315\314L?\315\314L?\253\252\*?", 16) = 16

13590 exit\_group(0) = ?

**13590 +++ exited with 0 +++**

13589 <... wait4 resumed>[{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 13590

**13589 --- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=13590, si\_uid=0, si\_status=0, si\_utime=1 /\* 0.01 s \*/, si\_stime=1 /\* 0.01 s \*/} ---**

13589 read(5, "\0\0\200?\315\314L?\315\314L?\253\252\*?", 16) = 16

13589 write(1, "\320\240\320\265\320\267\321\203\320\273\321\214\321\202\320\260\321\202 \320\264\320\265\320\273\320\265\320\275\320\270\321"..., 45) = 45

**13589 close(4) = 0**

**13589 close(5) = 0**

13589 lseek(0, -1, SEEK\_CUR) = -1 ESPIPE (Illegal seek)

13589 exit\_group(0) = ?

**13589 +++ exited with 0 +++**

**Вывод**

Лабораторная работа оказалась очень интересной и познавательной. Я познакомился с множеством утилит, а также разобрался в межпроцессорных взаимодействиях. Конечно, было нелегко, на лабораторную работу ушло немало времени. Наибольшую сложность вызывала необходимость пользоваться не знакомыми ранее библиотеками и понятиями.