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-210Б-23

Студент: Шведов А.И.

Преподаватель: Бахарев В.Д.

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

Дата: 03.10.24

Москва, 2024

**Постановка задачи**

**Вариант 3:**

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 int. 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 пайп и помещает дескрипторы в fd[0], fd[1], для чтения и записи.
* int write(int fd, const void\* buff, int count); – записывает по дескриптору fd count байт из buff.
* void exit(int number); – вызывает нормальное завершение программы с кодом number.
* int dup2(int fd1, int fd2); – делает эквиваентными дескрипторы fd1 и fd2.
* int exec(char\* path, const char\* argc); – заменяет текущий процесс на процесс path, с аргументами argc;
* int close(int fd); – закрывает дескриптор fd.
* pid\_t wait(int status) — функция, которая приостанавливает выполнение текущего процесса до тех пор, пока дочерний процесс не завершится,

Я создал два файла parent и child.

Программа создает неименованный канал (pipe) для передачи данных между родительским и дочерним процессами. Она запрашивает у пользователя ввод имени файла, после чего создаёт дочерний процесс с помощью fork(). Дочерний процесс перенаправляет стандартный ввод на чтение из канала и запускает другую программу (дочернюю программу), передавая ей имя файла. Родительский процесс читает данные из стандартного ввода до тех пор, пока пользователь не введёт пустую строку (нажатием Enter), и записывает эти данные в канал. После завершения записи родительский процесс закрывает канал и ожидает завершения дочернего процесса, предотвращая возникновение "зомби"-процессов.

В файле child я обрабатываю получнные из родительского процесса данные и записываю их в файл, а тажке проверяю деление на 0, и ввод пустой строки.

**Код программы**

**Parent.c**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <fcntl.h>

#include <sys/wait.h>

#define BUFFER\_SIZE 2048

int main()

{

    int pipe1[2];

    pid\_t pid;

    char buffer[BUFFER\_SIZE];

    char filename[100];

    // Создание pipe1

    if (pipe(pipe1) == -1)

    {

        perror("pipe");

        exit(1);

    }

    printf("Введите имя файла: ");

    scanf("%s", filename);

    // Создание дочернего процесса

    pid = fork();

    if (pid == -1)

    {

        perror("fork");

        exit(1);

    }

    if (pid == 0)

    {

        // Дочерний процесс

        // Закрытие ненужных сторон pipe'ов

        close(pipe1[1]);

        // Перенаправление стандартного ввода на pipe1

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

        close(pipe1[0]);

        // Запуск дочерней программы

        execlp("./child", "child", filename, NULL);

        perror("execlp");

        exit(1);

    }

    else

    {

        // Родительский процесс

        // Закрытие ненужных сторон pipe'ов

        close(pipe1[0]);

        ssize\_t bytes;

        while (bytes = read(STDIN\_FILENO, buffer, sizeof(buffer)))

        {

            if (bytes < 0)

            {

                const char msg[] = "error: failed to read from stdin\n";

                write(STDERR\_FILENO, msg, sizeof(msg));

                exit(EXIT\_FAILURE);

            }

            else if (buffer[0] == '\n')

            {

                break;

            }

            write(pipe1[1], buffer, strlen(buffer));

        }

        close(pipe1[1]);

        // Ожидание завершения дочернего процесса

        wait(NULL);

    }

    return 0;

}

**Child.c**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#define BUFFER\_SIZE 2024

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

    if (argc != 2) {

        fprintf(stderr, "Usage: %s <filename>\n", argv[0]);

        exit(1);

    }

    char \*filename = argv[1];

    FILE \*file = fopen(filename, "w");

    if (!file) {

        perror("fopen");

        exit(1);

    }

    char buffer[BUFFER\_SIZE];

    while (fgets(buffer, BUFFER\_SIZE, stdin)) {

        char \*token = strtok(buffer, " ");

        if (token == NULL) {

            continue;

        }

        double first\_num = atoi(token);

        int division\_by\_zero = 0;

        char result[BUFFER\_SIZE];

        snprintf(result, BUFFER\_SIZE, "%f", first\_num); // Записываем первое число в результат

        token = strtok(NULL, " ");

        while (token != NULL) {

            int num = atoi(token);

            if (num == 0) {

                division\_by\_zero = 1;

                break;

            }

            first\_num /= num;

            snprintf(result + strlen(result), BUFFER\_SIZE - strlen(result), " / %d", num);

            token = strtok(NULL, " ");

        }

        if (division\_by\_zero) {

            fprintf(stderr, "Error: Division by zero detected. Terminating processes.\n");

            fclose(file);

            exit(1); // Завершаем дочерний процесс

        }

        fprintf(file, "Result: %f\n", first\_num);

    }

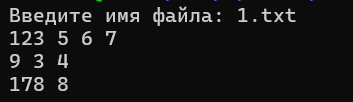
    fclose(file);

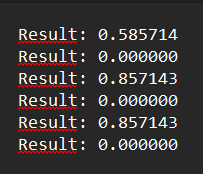
    return 0;

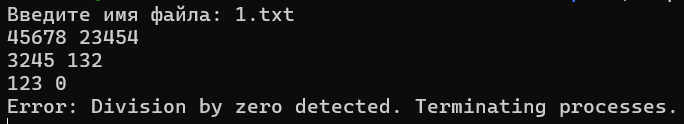
}

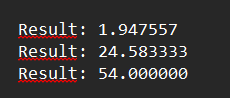
**Протокол работы программы**

Тестирование:





****

****

**Strace:**

execve("./parent", ["./parent"], 0x7fffcdc3f120 /\* 27 vars \*/) = 0

brk(NULL) = 0x5641f10db000

arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7ffde5529d50) = -1 EINVAL (Invalid argument)

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7fd5ae192000

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=37207, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 37207, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7fd5ae188000

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) = 0x7fd5adf5f000

mprotect(0x7fd5adf87000, 2023424, PROT\_NONE) = 0

mmap(0x7fd5adf87000, 1658880, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x7fd5adf87000

mmap(0x7fd5ae11c000, 360448, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1bd000) = 0x7fd5ae11c000

mmap(0x7fd5ae175000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x215000) = 0x7fd5ae175000

mmap(0x7fd5ae17b000, 52816, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7fd5ae17b000

close(3) = 0

mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7fd5adf5c000

arch\_prctl(ARCH\_SET\_FS, 0x7fd5adf5c740) = 0

set\_tid\_address(0x7fd5adf5ca10) = 879

set\_robust\_list(0x7fd5adf5ca20, 24) = 0

rseq(0x7fd5adf5d0e0, 0x20, 0, 0x53053053) = 0

mprotect(0x7fd5ae175000, 16384, PROT\_READ) = 0

mprotect(0x5641ef58f000, 4096, PROT\_READ) = 0

mprotect(0x7fd5ae1cc000, 8192, PROT\_READ) = 0

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

munmap(0x7fd5ae188000, 37207) = 0

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

newfstatat(1, "", {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0), ...}, AT\_EMPTY\_PATH) = 0

getrandom("\x39\x75\xca\xa0\x21\x35\x1c\x60", 8, GRND\_NONBLOCK) = 8

brk(NULL) = 0x5641f10db000

brk(0x5641f10fc000) = 0x5641f10fc000

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

**write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \320\270\320\274\321\217 \321\204\320\260\320\271\320\273\320\260"..., 34Введите имя файла: ) = 34**

**read(0, 1.txt**

**"1.txt\n", 1024) = 6**

**clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7fd5adf5ca10) = 904**

**close(3) = 0**

**read(0, 11 11 23**

**"11 11 23\n", 2048) = 9**

**write(4, "11 11 23\n\352\34\256\325\177", 14) = 14**

**read(0, 11 11 45**

**"11 11 45\n", 2048) = 9**

**write(4, "11 11 45\n\352\34\256\325\177", 14) = 14**

**read(0, 3 5 6**

**"3 5 6\n", 2048) = 6**

**write(4, "3 5 6\n45\n\352\34\256\325\177", 14) = 14**

**read(0,**

**"\n", 2048) = 1**

**close(4) = 0**

**wait4(-1, NULL, 0, NULL) = 904**

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=904, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=0} ---

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

**exit\_group(0) = ?**

+++ exited with 0 +++

**Вывод**

Было интересно решать лабораторную работу. Я научился использовать некоторые системные вызовы, а также обмениваться данными между процессами с помощью каналов. Было интересно узнать как можно писать программы используя их. Возникли трудности с обработкой всех ошибок системных вызовов в программе.