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.

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 <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];

    ssize\_t bytes\_read, bytes\_written;

    // Создание pipe1

    if (pipe(pipe1) == -1)

    {

        perror("pipe");

        exit(EXIT\_FAILURE);

    }

    // Сообщение пользователю для ввода имени файла

    const char \*msg = "Введите имя файла: ";

    bytes\_written = write(STDOUT\_FILENO, msg, strlen(msg));

    if (bytes\_written == -1)

    {

        const char error\_msg[] = "error: failed to write to stdout\n";

        write(STDERR\_FILENO, error\_msg, sizeof(error\_msg) - 1);

        exit(EXIT\_FAILURE);

    }

    // Чтение имени файла с консоли

    bytes\_read = read(STDIN\_FILENO, filename, sizeof(filename) - 1);

    if (bytes\_read == -1)

    {

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

        write(STDERR\_FILENO, error\_msg, sizeof(error\_msg) - 1);

        exit(EXIT\_FAILURE);

    }

    // Удаление символа новой строки, если он был введен

    if (bytes\_read > 0 && filename[bytes\_read - 1] == '\n')

    {

        filename[bytes\_read - 1] = '\0';

    }

    else

    {

        filename[bytes\_read] = '\0';

    }

    // Создание дочернего процесса

    const pid\_t child = fork();

    if (child == -1)

    {

        perror("fork");

        exit(EXIT\_FAILURE);

    }

    if (child == 0)

    {

        // Дочерний процесс

        pid\_t pid = getpid();

        {

            char msg[64];

            const int32\_t length = snprintf(msg, sizeof(msg), "%d: I'm a child\n", pid);

            write(STDOUT\_FILENO, msg, length);

        }

        close(pipe1[1]);

        // Перенаправление стандартного ввода на pipe1

        if (dup2(pipe1[0], STDIN\_FILENO) == -1)

        {

            perror("dup2");

            exit(1);

        }

        close(pipe1[0]);

        // Запуск дочерней программы

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

        perror("execlp");

        exit(1);

    }

    else

    {

        // Родительский процесс

        // Закрытие ненужной стороны pipe'а

        close(pipe1[0]);

        pid\_t pid = getpid();

            {

                char msg[64];

                const int32\_t length =

                    snprintf(msg, sizeof(msg), "%d: I'm a parent, my child has PID %d\n", pid, child);

                write(STDOUT\_FILENO, msg, length);

            }

        // Чтение данных с консоли и запись в pipe

        while ((bytes\_read = read(STDIN\_FILENO, buffer, sizeof(buffer))) > 0)

        {

            if (buffer[0] == '\n')

            {

                break;

            }

            bytes\_written = write(pipe1[1], buffer, bytes\_read);

            if (bytes\_written == -1)

            {

                const char error\_msg[] = "error: failed to write to pipe\n";

                write(STDERR\_FILENO, error\_msg, sizeof(error\_msg) - 1);

                exit(EXIT\_FAILURE);

            }

        }

        if (bytes\_read == -1)

        {

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

            write(STDERR\_FILENO, error\_msg, sizeof(error\_msg) - 1);

            exit(EXIT\_FAILURE);

        }

        // Закрытие pipe'а после записи

        close(pipe1[1]);

        // Ожидание завершения дочернего процесса

        wait(NULL);

    }

    return 0;

}

**Child.c**

#include <fcntl.h>

#include <stdbool.h>

#include <stdint.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <ctype.h>

#define BUFFER\_SIZE 2048

#define MAX\_NUMBERS 100

void convertStringToIntArray(const char \*str, int intArray[], int \*size)

{

    int i = 0, num = 0;

    \*size = 0;

    while (str[i] != '\n')

    {

        while (str[i] != '\n' && isspace(str[i]))

        {

            i++;

        }

        if (str[i] == '\n')

        {

            break;

        }

        num = 0;

        while (str[i] != '\n' && isdigit(str[i]))

        {

            num = num \* 10 + (str[i] - '0');

            i++;

        }

        if (\*size < MAX\_NUMBERS)

        {

            intArray[\*size] = num;

            (\*size)++;

        }

    }

} // Перевод строки в инт

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

{

    pid\_t pid = getpid();

    if (argc != 2)

    {

        const char msg[] = "Usage: <program> <filename>\n";

        write(STDERR\_FILENO, msg, sizeof(msg) - 1);

        exit(EXIT\_FAILURE);

    }

    char \*filename = argv[1];

    int file = open(filename, O\_WRONLY | O\_CREAT | O\_TRUNC, 0644);

    if (file == -1)

    {

        perror("open");

        exit(EXIT\_FAILURE);

    } // Открываем файл

    char buffer[BUFFER\_SIZE];

    ssize\_t bytes\_read;

    {

        char msg[128];

        int32\_t len =

            snprintf(msg, sizeof(msg) - 1,

                     "%d: Start typing lines of text. Press 'Ctrl-D' or 'Enter' with no input to exit\n", pid);

        write(STDOUT\_FILENO, msg, len);

    }

    while ((bytes\_read = read(STDIN\_FILENO, buffer, sizeof(buffer))) > 0)

    {

        if (buffer[0] == '\n')

        {

            break;

        }

        int array[MAX\_NUMBERS];

        int size = 0;

        convertStringToIntArray(buffer, array, &size);

        if (size < 2)

        {

            const char msg[] = "You have written few numbers\n";

            int32\_t written = write(STDOUT\_FILENO, msg, sizeof(msg));

            if (written != sizeof(msg))

            {

                const char msg[] = "error: failed to write to file\n";

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

                exit(EXIT\_FAILURE);

            }

        } // Обработка малого количества аргументов

        int divisible = array[0];

        for (int i = 1; i != size; ++i)

        {

            if (array[i] == 0)

            {

                const char ermsg[] = "Division by zero\n";

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

                exit(EXIT\_FAILURE);

            }

            char msg[32];

            int32\_t len = snprintf(msg, sizeof(msg) - 1, "%d : %d = %lf\n", divisible, array[i],

                                   ((float)divisible / array[i]));

            int32\_t written = write(file, msg, len);

            if (written != len)

            {

                const char msg[] = "error: failed to write to file\n";

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

                exit(EXIT\_FAILURE);

            }

        }

    }

    if (bytes\_read == -1)

    {

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

        write(STDERR\_FILENO, msg, sizeof(msg) - 1);

        close(file);

        exit(1);

    }

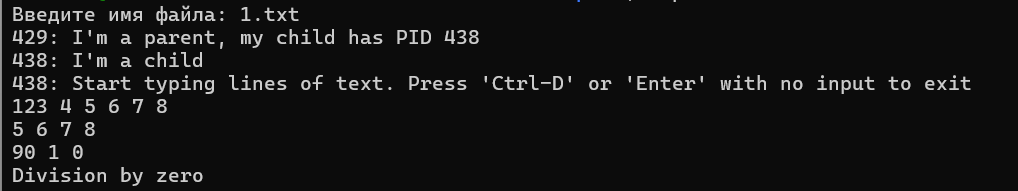
    close(file);

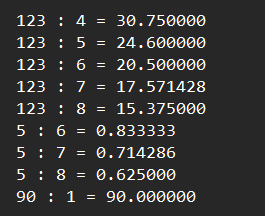
    return 0;

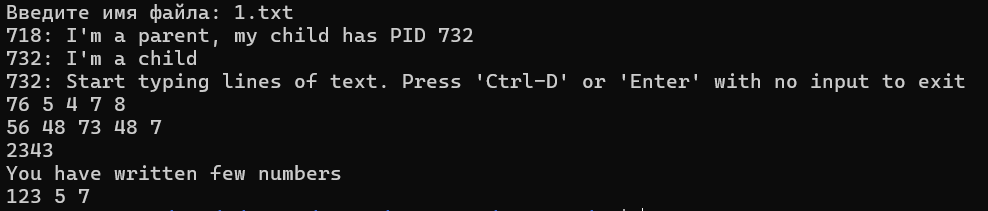
}

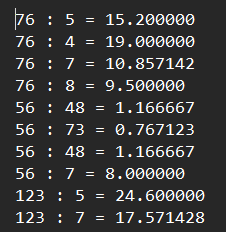
**Протокол работы программы**

Тестирование:



****

****

****

**Strace:**

execve("./parent", ["./parent"], 0x7ffcd9f5a690 /\* 27 vars \*/) = 0

brk(NULL) = 0x55c0685c1000

arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7ffc92b46dd0) = -1 EINVAL (Invalid argument)

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7fd8252cd000

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) = 0x7fd8252c3000

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) = 0x7fd82509a000

mprotect(0x7fd8250c2000, 2023424, PROT\_NONE) = 0

mmap(0x7fd8250c2000, 1658880, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x7fd8250c2000

mmap(0x7fd825257000, 360448, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1bd000) = 0x7fd825257000

mmap(0x7fd8252b0000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x215000) = 0x7fd8252b0000

mmap(0x7fd8252b6000, 52816, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7fd8252b6000

close(3) = 0

mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7fd825097000

arch\_prctl(ARCH\_SET\_FS, 0x7fd825097740) = 0

set\_tid\_address(0x7fd825097a10) = 1139

set\_robust\_list(0x7fd825097a20, 24) = 0

rseq(0x7fd8250980e0, 0x20, 0, 0x53053053) = 0

mprotect(0x7fd8252b0000, 16384, PROT\_READ) = 0

mprotect(0x55c066eda000, 4096, PROT\_READ) = 0

mprotect(0x7fd825307000, 8192, PROT\_READ) = 0

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

munmap(0x7fd8252c3000, 37207) = 0

**pipe2([3, 4], 0) = 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", 99) = 6**

**clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7fd825097a10) = 1157**

**1157: I'm a child**

close(3) = 0

**getpid() = 1139**

**write(1, "1139: I'm a parent, my child has"..., 421139: I'm a parent, my child has PID 1157**

**) = 42**

**read(0, 1157: Start typing lines of text. Press 'Ctrl-D' or 'Enter' with no input to exit**

**123 45 7 8 9**

**"123 45 7 8 9\n", 2048) = 13**

**write(4, "123 45 7 8 9\n", 13) = 13**

**read(0, 12 35 67 9**

**"12 35 67 9\n", 2048) = 11**

**write(4, "12 35 67 9\n", 11) = 11**

**read(0, 23 5**

**"23 5\n", 2048) = 5**

**write(4, "23 5\n", 5) = 5**

**read(0, "", 2048) = 0**

close(4) = 0

**wait4(-1, NULL, 0, NULL) = 1157**

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=1157, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=0} ---

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.