**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ  
ФЕДЕРАЦИИ МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ  
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)**

**ЛАБОРАТОРНАЯ РАБОТА №2**

**по курсу операционные системы I семестр, 2021/22  
уч. год**

Студент: Белоусов Егор Владимирович

Группа: М8О-207Б-20

Преподаватель: Миронов Е. С.

Вариант: 3

Дата:

Оценка:

**Цель работы:**

Приобретение практических навыков в:

* Управление процессами в ОС
* Обеспечение обмена данных между процессами посредством каналов

**Введение**

Данную лабораторную работу я выполнял в среде UNIX.

Процесс – это абстракция, описывающая выполняющую программу. В моем задании требуется 2 процесса: родительский и дочерний. Взаимодействие между процессами осуществляется с помощью каналов pipe1 и pipe2. Процессор-родитель считывает файл, в который нужно выгрузить результат, а также считывает числа, перенаправляя свой поток входных данных через pipe1, дочерний процесс соответственно считывает данные через pipe1. Затем выполняется само задание: для заданной последовательности чисел в выходной файл выводится первое число разделенное на все последующие.

В конце закрываются все дескрипторы.  
  
**Системные вызовы:**

**int putchar(char c)** – записывает символ c в стандартный поток вывода, а возвращает записанный символ или EOF, в случае ошибки.

**pid\_t fork(void)** – создает дочерний процесс. Если возвращает 0, то созданный текущий процесс – ребенок, если >0, то – родитель, если <0, то – ошибка(и текущий процесс – родитель).

**exit(int status)** – выходит из процесса с заданным статусом.

**pid\_t wait(int \*status)** - приостанавливает выполнение текущего процесса до тех пор, пока дочерний процесс не завершится, или до появления сигнала, который либо завершает текущий процесс, либо требует вызвать функцию-обработчик.

**int pipe(int \*fd);** – предоставляет средства передачи данных между двумя процессами(неименованный канал).

**int close(int fd)** – закрывает файловый дескриптор.

**int read(int fd, void \*buffer, int nbyte)** – читает nbyte.

**ssize\_t write(int** fd**, const void \***buf**, size\_t** count**);** - записывает *count* байтов из буфера начиная с *buf* в файл с файловым дескриптором *fd*.

**2. Исходный код**

**parent.c**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

#include <sys/wait.h>

int main(){

    char \*fileName = NULL;

    size\_t length;

    getline(&fileName,&length, stdin);

    fileName[strlen(fileName) - 1] = '\0';

    int fd[2];

    int d[2];

    if(pipe(fd) || pipe(d)){

        perror("Pipe error\n");

        exit(1);

    }

    int rfd = fd[0], wfd = fd[1];

    int rd = d[0], wd = d[1];

    pid\_t pid = fork();

    if(pid == -1){

        perror("Fork error\n");

        exit(1);

    }

    else if(!pid){

        close(rd);

        close(wfd);

        if(dup2(wd, STDOUT\_FILENO) == -1){

            perror("Dup2 error\n");

            exit(1);

        }

        close(wd);

        if(dup2(rfd, STDIN\_FILENO) == -1){

            perror("Dup2 error\n");

            exit(1);

        }

        close(rfd);

        execl("child", "child", fileName, NULL);

        perror("execl");

        exit(1);

    }

    char c;

    while(scanf("%c",&c) != EOF){

        write(wfd, &c, sizeof(c));

    }

    close(wfd);

    close(rfd);

    close(wd);

    while(read(rd,&c,sizeof(char)) > 0){

        putchar(c);

    }

    close(rd);

    int status;

    if(wait(&status) == -1){

        perror("wait");

    }

    if(!WIFEXITED(status) || (WIFEXITED(status) && (WEXITSTATUS(status)) != 0)){

        fprintf(stderr, "Error in child process!\n");

        return 1;

    }

    return 0;

}

**child.c**

#include "stdio.h"

#include "stdlib.h"

#include <unistd.h>

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

    char \* fileName = argv[1];

    double res = 0.0;

    int x;

    int flag = 1;

    char c;

    FILE \* out = fopen(fileName, "w");

    if(out == NULL){

        perror("File not opened\n");

        exit(1);

    }

    while(scanf("%d%c", &x, &c) != EOF) {

        if(flag){

            res = x;

            flag = 0;

        }

        else {

            if(x == 0){

                printf("Zero division!\n");

                exit(1);

            }

            res /= (double)x;

        }

        if(c == '\n') {

            fprintf(out,"%f\n", res);

            res = 0;

            flag = 1;

            continue;

        }

    }

    write(STDOUT\_FILENO, "Child finished successfully!\n",sizeof("Child finished successfully!\n"));

    fclose(out);

    return 0;

}

**3. Пример работы**

egorb@LAPTOP-TIFMO0IA:/mnt/d/os/lab2$ gcc child.c -o child

egorb@LAPTOP-TIFMO0IA:/mnt/d/os/lab2$ gcc parent.c -o parent

egorb@LAPTOP-TIFMO0IA:/mnt/d/os/lab2$ ./parent

out.txt

1 2

2 3

Child finished successfully!

egorb@LAPTOP-TIFMO0IA:/mnt/d/os/lab2$ cat out.txt

0.500000

0.666667

egorb@LAPTOP-TIFMO0IA:/mnt/d/os/lab2$ ./parent

test.txt

1 0

Zero division!

Error in child process!

egorb@LAPTOP-TIFMO0IA:/mnt/d/os/lab2$ cat out.txt

0.500000

0.666667

egorb@LAPTOP-TIFMO0IA:/mnt/d/os/lab2$ cat test.txt

**Strace**

В данной лабораторной работе применяется множество системных вызовов: *fopen, dup2, fork, pipe, execl*. strace показал всю статистику по данным системным вызовам. Так же показана диагностика для дочернего процесса (при этом в начале строки указывается pid).

*egorb@LAPTOP-TIFMO0IA:/mnt/d/os/lab2$ strace -f ./parent << EOF*

*> out.txt*

*> 2 4*

*> 8 6*

*> 8 9*

*> 2 0*

*> 3 5*

*> EOF*

*execve("./parent", ["./parent"], 0x7ffd922c0228 /\* 29 vars \*/) = 0*

*brk(NULL) = 0x55f9947de000*

*arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7ffea384ac70) = -1 EINVAL (Invalid argument)*

*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*

*fstat(3, {st\_mode=S\_IFREG|0644, st\_size=34691, ...}) = 0*

*mmap(NULL, 34691, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f76dfc88000*

*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\0\360q\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\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0", 32, 848) = 32*

*pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\363\377?\332\200\270\27\304d\245n\355Y\377\t\334"..., 68, 880) = 68*

*fstat(3, {st\_mode=S\_IFREG|0755, st\_size=2029224, ...}) = 0*

*mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f76dfc86000*

*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\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0", 32, 848) = 32*

*pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\363\377?\332\200\270\27\304d\245n\355Y\377\t\334"..., 68, 880) = 68*

*mmap(NULL, 2036952, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f76dfa94000*

*mprotect(0x7f76dfab9000, 1847296, PROT\_NONE) = 0*

*mmap(0x7f76dfab9000, 1540096, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x25000) = 0x7f76dfab9000*

*mmap(0x7f76dfc31000, 303104, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x19d000) = 0x7f76dfc31000*

*mmap(0x7f76dfc7c000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1e7000) = 0x7f76dfc7c000*

*mmap(0x7f76dfc82000, 13528, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f76dfc82000*

*close(3) = 0*

*arch\_prctl(ARCH\_SET\_FS, 0x7f76dfc87540) = 0*

*mprotect(0x7f76dfc7c000, 12288, PROT\_READ) = 0*

*mprotect(0x55f9946fc000, 4096, PROT\_READ) = 0*

*mprotect(0x7f76dfcbe000, 4096, PROT\_READ) = 0*

*munmap(0x7f76dfc88000, 34691) = 0*

*brk(NULL) = 0x55f9947de000*

*brk(0x55f9947ff000) = 0x55f9947ff000*

*fstat(0, {st\_mode=S\_IFREG|0400, st\_size=28, ...}) = 0*

*read(0, "out.txt\n2 4\n8 6\n8 9\n2 0\n3 5\n", 4096) = 28*

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

*pipe([5, 6]) = 0*

*clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLDstrace: Process 2114 attached*

*, child\_tidptr=0x7f76dfc87810) = 2114*

*[pid 2114] close(5 <unfinished ...>*

*[pid 2113] write(4, "2", 1 <unfinished ...>*

*[pid 2114] <... close resumed>) = 0*

*[pid 2113] <... write resumed>) = 1*

*[pid 2114] close(4 <unfinished ...>*

*[pid 2113] write(4, " ", 1 <unfinished ...>*

*[pid 2114] <... close resumed>) = 0*

*[pid 2113] <... write resumed>) = 1*

*[pid 2114] dup2(6, 1 <unfinished ...>*

*[pid 2113] write(4, "4", 1 <unfinished ...>*

*[pid 2114] <... dup2 resumed>) = 1*

*[pid 2113] <... write resumed>) = 1*

*[pid 2114] close(6 <unfinished ...>*

*[pid 2113] write(4, "\n", 1 <unfinished ...>*

*[pid 2114] <... close resumed>) = 0*

*[pid 2113] <... write resumed>) = 1*

*[pid 2114] dup2(3, 0 <unfinished ...>*

*[pid 2113] write(4, "8", 1 <unfinished ...>*

*[pid 2114] <... dup2 resumed>) = 0*

*[pid 2113] <... write resumed>) = 1*

*[pid 2114] close(3 <unfinished ...>*

*[pid 2113] write(4, " ", 1 <unfinished ...>*

*[pid 2114] <... close resumed>) = 0*

*[pid 2113] <... write resumed>) = 1*

*[pid 2114] execve("child", ["child", "out.txt"], 0x7ffea384ad58 /\* 29 vars \*/ <unfinished ...>*

*[pid 2113] write(4, "6", 1) = 1*

*[pid 2113] write(4, "\n", 1) = 1*

*[pid 2113] write(4, "8", 1) = 1*

*[pid 2113] write(4, " ", 1) = 1*

*[pid 2113] write(4, "9", 1) = 1*

*[pid 2113] write(4, "\n", 1) = 1*

*[pid 2113] write(4, "2", 1) = 1*

*[pid 2113] write(4, " ", 1) = 1*

*[pid 2113] write(4, "0", 1) = 1*

*[pid 2113] write(4, "\n", 1) = 1*

*[pid 2113] write(4, "3", 1) = 1*

*[pid 2113] write(4, " ", 1) = 1*

*[pid 2113] write(4, "5", 1) = 1*

*[pid 2113] write(4, "\n", 1) = 1*

*[pid 2113] read(0, "", 4096) = 0*

*[pid 2113] close(4) = 0*

*[pid 2113] close(3) = 0*

*[pid 2113] close(6) = 0*

*[pid 2113] read(5, <unfinished ...>*

*[pid 2114] <... execve resumed>) = 0*

*[pid 2114] brk(NULL) = 0x556e001af000*

*[pid 2114] arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7fff189582f0) = -1 EINVAL (Invalid argument)*

*[pid 2114] access("/etc/ld.so.preload", R\_OK) = -1 ENOENT (No such file or directory)*

*[pid 2114] openat(AT\_FDCWD, "/etc/ld.so.cache", O\_RDONLY|O\_CLOEXEC) = 3*

*[pid 2114] fstat(3, {st\_mode=S\_IFREG|0644, st\_size=34691, ...}) = 0*

*[pid 2114] mmap(NULL, 34691, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f57eed34000*

*[pid 2114] close(3) = 0*

*[pid 2114] openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libc.so.6", O\_RDONLY|O\_CLOEXEC) = 3*

*[pid 2114] read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\360q\2\0\0\0\0\0"..., 832) = 832*

*[pid 2114] 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*

*[pid 2114] pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0", 32, 848) = 32*

*[pid 2114] pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\363\377?\332\200\270\27\304d\245n\355Y\377\t\334"..., 68, 880) = 68*

*[pid 2114] fstat(3, {st\_mode=S\_IFREG|0755, st\_size=2029224, ...}) = 0*

*[pid 2114] mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f57eed32000*

*[pid 2114] 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*

*[pid 2114] pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0", 32, 848) = 32*

*[pid 2114] pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\363\377?\332\200\270\27\304d\245n\355Y\377\t\334"..., 68, 880) = 68*

*[pid 2114] mmap(NULL, 2036952, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f57eeb40000*

*[pid 2114] mprotect(0x7f57eeb65000, 1847296, PROT\_NONE) = 0*

*[pid 2114] mmap(0x7f57eeb65000, 1540096, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x25000) = 0x7f57eeb65000*

*[pid 2114] mmap(0x7f57eecdd000, 303104, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x19d000) = 0x7f57eecdd000*

*[pid 2114] mmap(0x7f57eed28000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1e7000) = 0x7f57eed28000*

*[pid 2114] mmap(0x7f57eed2e000, 13528, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f57eed2e000*

*[pid 2114] close(3) = 0*

*[pid 2114] arch\_prctl(ARCH\_SET\_FS, 0x7f57eed33540) = 0*

*[pid 2114] mprotect(0x7f57eed28000, 12288, PROT\_READ) = 0*

*[pid 2114] mprotect(0x556dff77a000, 4096, PROT\_READ) = 0*

*[pid 2114] mprotect(0x7f57eed6a000, 4096, PROT\_READ) = 0*

*[pid 2114] munmap(0x7f57eed34000, 34691) = 0*

*[pid 2114] brk(NULL) = 0x556e001af000*

*[pid 2114] brk(0x556e001d0000) = 0x556e001d0000*

*[pid 2114] openat(AT\_FDCWD, "out.txt", O\_WRONLY|O\_CREAT|O\_TRUNC, 0666) = 3*

*[pid 2114] fstat(0, {st\_mode=S\_IFIFO|0600, st\_size=0, ...}) = 0*

*[pid 2114] read(0, "2 4\n8 6\n8 9\n2 0\n3 5\n", 4096) = 20*

*[pid 2114] fstat(3, {st\_mode=S\_IFREG|0777, st\_size=0, ...}) = 0*

*[pid 2114] fstat(1, {st\_mode=S\_IFIFO|0600, st\_size=0, ...}) = 0*

*[pid 2114] write(3, "0.500000\n1.333333\n0.888889\n", 27) = 27*

*[pid 2114] write(1, "Zero division!\n", 15 <unfinished ...>*

*[pid 2113] <... read resumed>"Z", 1) = 1*

*[pid 2114] <... write resumed>) = 15*

*[pid 2113] fstat(1, <unfinished ...>*

*[pid 2114] lseek(0, -4, SEEK\_CUR <unfinished ...>*

*[pid 2113] <... fstat resumed>{st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0x1), ...}) = 0*

*[pid 2114] <... lseek resumed>) = -1 ESPIPE (Illegal seek)*

*[pid 2113] read(5, <unfinished ...>*

*[pid 2114] exit\_group(1 <unfinished ...>*

*[pid 2113] <... read resumed>"e", 1) = 1*

*[pid 2114] <... exit\_group resumed>) = ?*

*[pid 2113] read(5, "r", 1) = 1*

*[pid 2113] read(5, "o", 1) = 1*

*[pid 2113] read(5, " ", 1) = 1*

*[pid 2113] read(5, "d", 1) = 1*

*[pid 2113] read(5, "i", 1) = 1*

*[pid 2113] read(5, "v", 1) = 1*

*[pid 2113] read(5, "i", 1) = 1*

*[pid 2113] read(5, "s", 1) = 1*

*[pid 2114] +++ exited with 1 +++*

*--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=2114, si\_uid=1000, si\_status=1, si\_utime=0, si\_stime=0} ---*

*read(5, "i", 1) = 1*

*read(5, "o", 1) = 1*

*read(5, "n", 1) = 1*

*read(5, "!", 1) = 1*

*read(5, "\n", 1) = 1*

*write(1, "Zero division!\n", 15Zero division!*

*) = 15*

*read(5, "", 1) = 0*

*close(5) = 0*

*wait4(-1, [{WIFEXITED(s) && WEXITSTATUS(s) == 1}], 0, NULL) = 2114*

*write(2, "Error in child process!\n", 24Error in child process!*

*) = 24*

*exit\_group(1) = ?*

*+++ exited with 1 +++*

Отдельно статистика по системным вызовам:

*egorb@LAPTOP-TIFMO0IA:/mnt/d/os/lab2$ strace -c ./parent << EOF*

*> out.txt*

*> 2 4*

*> 8 6*

*> 8 9*

*> 2 0*

*> 3 5*

*> EOF*

*Zero division!*

*Error in child process!*

*% time seconds usecs/call calls errors syscall*

*------ ----------- ----------- --------- --------- ----------------*

*0.00 0.000000 0 19 read*

*0.00 0.000000 0 22 write*

*0.00 0.000000 0 6 close*

*0.00 0.000000 0 4 fstat*

*0.00 0.000000 0 7 mmap*

*0.00 0.000000 0 4 mprotect*

*0.00 0.000000 0 1 munmap*

*0.00 0.000000 0 3 brk*

*0.00 0.000000 0 6 pread64*

*0.00 0.000000 0 1 1 access*

*0.00 0.000000 0 2 pipe*

*0.00 0.000000 0 1 clone*

*0.00 0.000000 0 1 execve*

*0.00 0.000000 0 1 wait4*

*0.00 0.000000 0 2 1 arch\_prctl*

*0.00 0.000000 0 2 openat*

*------ ----------- ----------- --------- --------- ----------------*

*100.00 0.000000 82 2 total*

**4. Выводы**

Данная лабораторная работа научила меня работать с таким важнейшим понятием, как **процессы**. Я научился создавать дочерние процессы в системе UNIX используя **fork**. При этом мне потребовалось распределять потоки ввода и вывода между ними используя **pipe**. Также я использовал различные системные вызовы (**dup2, fopen**) и разобрался с файловыми деструкторами. Используя процессы можно разбить программу на более мелкие части, что делает работу с программой более удобной и структурированной.