

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-214БВ-23

Студент: Богданов М.А.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 28.09.24

Москва, 2024

Постановка задачи

Вариант 4.

Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса пишет имя файла, которое будет передано при создании дочернего процесса. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс передает команды пользователя через `ripe1`, который связан с стандартным входным потоком дочернего процесса. Дочерний процесс при необходимости передает данные в родительский процесс через `ripe2`. Результаты своей работы дочерний процесс пишет в созданный им файл. Допускается просто открыть файл и писать туда, не перенаправляя стандартный поток вывода.

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();` – создает дочерний процесс.

`int pipe();` – создает каналы для взаимодействия между процессами

`dup2();` – перенаправляет стандартные потоки

`execv();` – запускает новую программу

`read();` – читает данные

Программа создает два `ripe` – канала для взаимодействия родительского и дочернего процесса. Далее запускается дочерний процесс через `fork` и `execv`. Далее пользователь вводит имя файла для отладки, который передается дочернему процессу. Второй и последующими строками пользователь вводит числа типа `float` в командную строку разделяя их пробелами. Чисел может быть неограниченное кол-во. Для разделения и подсчета результата реализована специальная функция `buf_to_numbers()`. При помощи этой функции дочерний процесс подсчитывает необходимое число и записывает результат в переданный файл. Так же существует обработка ошибок. При встрече любой ошибки при вычислениях дочерний процесс передает текст ошибки в родительский процесс, тот выводит ее в `stderr`, и затем оба процесса немедленно завершаются.

Код программы

lab1_client.c

```
#include <stdint.h>
#include <stdbool.h>

#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <string.h>
static char SERVER_PROGRAM_NAME[] = "lab1_server";

int main(int argc, char **argv) {
    char filename[256];
    ssize_t bytes_read;
    bytes_read = read(STDIN_FILENO, filename, sizeof(filename) - 1);
    filename[strcspn(filename, "\n")] = '\0';
    char progpath[1024];
    {
        ssize_t len = readlink("/proc/self/exe", progpath,
                               sizeof(progpath) - 1);
        if (len == -1) {
            const char msg[] = "error: failed to read full program path\n";
            write(STDERR_FILENO, msg, sizeof(msg));
            exit(EXIT_FAILURE);
        }
        while (progpath[len] != '/')
            --len;
        progpath[len] = '\0';
    }
    int client_to_server[2];
    if (pipe(client_to_server) == -1) {
        const char msg[] = "error: failed to create pipe\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }

    int server_to_client[2];
    if (pipe(server_to_client) == -1) {
        const char msg[] = "error: failed to create pipe\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }

    const pid_t child = fork();

    switch (child) {
```

```

case -1: {
    const char msg[] = "error: failed to spawn new process\n";
    write(STDERR_FILENO, msg, sizeof(msg));
    exit(EXIT_FAILURE);
} break;
case 0: {
    close(client_to_server[1]);
    close(server_to_client[0]);
    dup2(client_to_server[0], STDIN_FILENO);
    close(client_to_server[0]);
    dup2(server_to_client[1], STDOUT_FILENO);
    close(server_to_client[1]);
{
    char path[1024];
    strcpy(path, progpath);
    strcat(path, "/");
    strcat(path, SERVER_PROGRAM_NAME);
    char *const args[] = {SERVER_PROGRAM_NAME, filename, NULL};
    int32_t status = execv(path, args);
    if (status == -1) {
        const char msg[] = "error: failed to exec into new executable image\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }
}
} break;

```

```

default: {
    close(client_to_server[0]);
    close(server_to_client[1]);
    char buf[4096], err_buf[256];
    ssize_t bytes, err_bytes;
    while ((bytes = read(STDIN_FILENO, buf, sizeof(buf)))) {
        if (bytes < 0) {
            const char msg[] = "error: failed to read from stdin\n";
            write(STDERR_FILENO, msg, sizeof(msg));
            exit(EXIT_FAILURE);
        } else if (buf[0] == '\n') {
            break;
        }
        write(client_to_server[1], buf, bytes);
    }
    if ((err_bytes = read(server_to_client[0], err_buf, sizeof(err_buf)))) {
        if (err_buf[0] == 'e'){
            write(STDOUT_FILENO, err_buf, err_bytes);
            exit(EXIT_FAILURE);
        }
    }
}

close(client_to_server[1]);
close(server_to_client[0]);

```

```
    wait(NULL);
} break;
}
}
```

lab1_server.c

```
#include <stdint.h>
#include <stdbool.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>

float buf_to_numbers(char buf[]){

    float n = 10, num2 = 0, result;
    int num_path, i = 0, flag = 0, min_flag = 0, first_flag = 0;
    while (true){

        if (buf[i] != '1' && buf[i] != '2' && buf[i] != '3' && buf[i] != '4' && buf[i] != '5' \
            && buf[i] != '6' && buf[i] != '7' && buf[i] != '8' && buf[i] != '9' && buf[i] != '0' && buf[i] != '.' && buf[i] != ' ' \
            && buf[i] != '-' && buf[i] != '\n'){

            const char msg[] = "error: invalid data\n";
            write(STDOUT_FILENO, msg, sizeof(msg));
            exit(EXIT_FAILURE);
        }

        if (buf[i] == ' '){

            n = 10;
            flag = 0;
            if (min_flag == 1){
                num2 *= -1;
            }
            min_flag = 0;
            if (first_flag == 0){
                result = num2;
                first_flag = 1;
            }
        }

        else if (num2 != 0 && first_flag == 1){
            result /= num2;
        }

        else if (num2 == 0 && first_flag == 1){

            const char msg[] = "error: can't devide by zero\n";
            write(STDOUT_FILENO, msg, sizeof(msg));
            exit(EXIT_FAILURE);
        }

        num2 = 0;
    }

    else if (buf[i] == '\n' || buf[i] == '\0'){

        n = 10;
        flag = 0;
        if (min_flag == 1){
            num2 *= -1;
        }
        min_flag = 0;
        if (first_flag == 0){
            result = num2;
            first_flag = 1;
        }
    }
}
```

```

    }
    else if(num2 != 0 && first_flag == 1){
        result /= num2;
    }
    else if(num2 == 0 && first_flag == 1){
        const char msg[] = "error: can't devide by zero\n";
        write(STDOUT_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }
    num2 = 0;
    break;
}
else if(buf[i] == '.'){
    flag = 1;
    n = 0.1;
}
else if(buf[i] == '-'){
    min_flag = 1;
}
else if(flag == 0){
    num2 = (buf[i] - '0') + num2 * n;
}
else if(flag == 1){
    num2 += (buf[i] - '0') * n;
    n /= 10;
}
++i;
}
return result;
}

```

```

int main(int argc, char **argv) {
    char buf[4096];
    ssize_t bytes;
    int32_t file = open(argv[1], O_WRONLY | O_CREAT | O_TRUNC, 0600);
    if(file == -1) {
        const char msg[] = "error: failed to open requested file\n";
        write(STDOUT_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }

    while ((bytes = read(STDIN_FILENO, buf, sizeof(buf)))) {
        if(bytes < 0) {
            const char msg[] = "error: failed to read from stdin\n";
            write(STDOUT_FILENO, msg, sizeof(msg));
            exit(EXIT_FAILURE);
        }

        float result = buf_to_numbers(buf);
        bytes = sprintf(buf, "%4f\n", result);
    }
}

```

```

{
    int32_t written = write(file, buf, bytes);
    if(written != bytes) {
        const char msg[] = "error: failed to write to file\n";
        write(STDOUT_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }
}

const char msg[] = "success";
write(STDOUT_FILENO, msg, sizeof(msg));
}

if(bytes == 0) {
    const char term = '\0';
    write(file, &term, sizeof(term));
}

close(file);
}

```

Протокол работы программы

Тестирование:

#1

```

strace -o mmm.log ./lab1_client 111
2 2 2
1 1 1 1 1 1
23.32323 14 4
0.4 -3 4 1
121201921029 12121
ы
error: invalid data
bogdanoff@arch ~/$/O/lab1 (main) [1]> cat 111
0.5000
1.0000
0.4165
-0.0333
9999333.0000
#2

```

```

bogdanoff@arch ~/$/O/lab1 (main)> strace -o mmm.log ./lab1_client 111.txt
12 -2 2

```

1 0 2

error: can't devide by zero

bogdanoff@arch ~\$O/lab1 (main) [1]> cat 111.txt

-3.0000

bogdanoff@arch ~\$O/lab1 (main)>

Strace:

#1

```
execve("./lab1_client", ["./lab1_client"], 0x7ffe4e33b4a0 /* 33 vars */) = 0
brk(NULL) = 0x55b0efde5000
access("/etc/ld.so.preload", R_OK) = -1 ENOENT (Нет такого файла или каталога)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=102215, ...}) = 0
mmap(NULL, 102215, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f61ed03e000
close(3) = 0
openat(AT_FDCWD, "/usr/lib/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\0>\0\1\0\0\0000x\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"..., 896, 64) = 896
fstat(3, {st_mode=S_IFREG|0755, st_size=2149728, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f61ed03c000
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"..., 896, 64) = 896
mmap(NULL, 2174000, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f61ece00000
mmap(0x7f61ece24000, 1515520, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x24000) = 0x7f61ece24000
mmap(0x7f61ecf96000, 454656, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x196000) = 0x7f61ecf96000
mmap(0x7f61ed005000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x204000) = 0x7f61ed005000
mmap(0x7f61ed00b000, 31792, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f61ed00b000
close(3) = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f61ed039000
arch_prctl(ARCH_SET_FS, 0x7f61ed039740) = 0
set_tid_address(0x7f61ed039a10) = 121377
set_robust_list(0x7f61ed039a20, 24) = 0
rseq(0x7f61ed039680, 0x20, 0, 0x53053053) = 0
mprotect(0x7f61ed005000, 16384, PROT_READ) = 0
mprotect(0x55b0dec20000, 4096, PROT_READ) = 0
mprotect(0x7f61ed098000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
getrandom("\x61\x95\xc4\x11\x92\x6a\xc3\xd2", 8, GRND_NONBLOCK) = 8
munmap(0x7f61ed03e000, 102215) = 0
read(0, "111\n", 255) = 4
readlink("/proc/self/exe", "/home/bogdanoff/study/OS/lab1/la"..., 1023) = 41
pipe2([3, 5], 0) = 0
pipe2([6, 7], 0) = 0
rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7f61ed039a10) = 121384
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
close(3) = 0
close(7) = 0
read(0, "2 2 2\n", 4096) = 6
write(5, "2 2 2\n", 6) = 6
read(6, "success\0", 256) = 8
read(0, "1 1 1 1 1 1 1\n", 4096) = 14
```

```
write(5, "1 1 1 1 1 1\n", 14) = 14
read(6, "success\0", 256) = 8
read(0, "23.32323 14 4\n", 4096) = 14
write(5, "23.32323 14 4\n", 14) = 14
read(6, "success\0", 256) = 8
read(0, "0.4 -3 4 1\n", 4096) = 11
write(5, "0.4 -3 4 1\n", 11) = 11
read(6, "success\0", 256) = 8
read(0, "121201921029 12121\n", 4096) = 19
write(5, "121201921029 12121\n", 19) = 19
read(6, "success\0", 256) = 8
read(0, "\321\213\n", 4096) = 3
write(5, "\321\213\n", 3) = 3
read(6, "error: invalid data\n\0", 256) = 21
write(1, "error: invalid data\n\0", 21) = 21
exit_group(1) = ?
```

#2

```
execve("./lab1_client", ["../lab1_client"], 0x7ffc5d86c070 /* 33 vars */ = 0
brk(NULL) = 0x561c1b489000
access("/etc/ld.so.preload", R_OK) = -1 ENOENT (Нет такого файла или каталога)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=102215, ...}) = 0
mmap(NULL, 102215, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f010ba90000
close(3) = 0
openat(AT_FDCWD, "/usr/lib/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "177ELF2\1\1\3\0\0\0\0\0\0\0\3\0>\0\1\0\0\0000x2\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\0\0\0...", 896, 64) = 896
fstat(3, {st_mode=S_IFREG|0755, st_size=2149728, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f010ba8e000
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\0\0\0...", 896, 64) = 896
mmap(NULL, 2174000, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f010b800000
mmap(0x7f010b824000, 1515520, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x24000) = 0x7f010b824000
mmap(0x7f010b996000, 454656, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x196000) = 0x7f010b996000
mmap(0x7f010ba05000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x204000) = 0x7f010ba05000
mmap(0x7f010ba0b000, 31792, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f010ba0b000
close(3) = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f010ba8b000
arch_prctl(ARCH_SET_FS, 0x7f010ba8b740) = 0
set_tid_address(0x7f010ba8ba10) = 121623
set_robust_list(0x7f010ba8ba20, 24) = 0
rseq(0x7f010ba8b680, 0x20, 0, 0x53053053) = 0
mprotect(0x7f010ba05000, 16384, PROT_READ) = 0
mprotect(0x561c185e5000, 4096, PROT_READ) = 0
mprotect(0x7f010baea000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
getrandom("\x57\x52\xc1\x0f\x3c\x26\xe6\x8a", 8, GRND_NONBLOCK) = 8
munmap(0x7f010ba90000, 102215) = 0
read(0, "111.txt\n", 255) = 8
readlink("/proc/self/exe", "/home/bogdanoff/study/OS/lab1/la...", 1023) = 41
pipe2([3, 5], 0) = 0
```

```
pipe2([6, 7], 0) = 0
rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7f010ba8ba10) = 121631
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
close(3) = 0
close(7) = 0
read(0, "12 -2 2\n", 4096) = 8
write(5, "12 -2 2\n", 8) = 8
read(6, "success\0", 256) = 8
read(0, "1 0 2\n", 4096) = 6
write(5, "1 0 2\n", 6) = 6
read(6, "error: can't devide by zero\n\0", 256) = 29
write(1, "error: can't devide by zero\n\0", 29) = 29
exit_group(1) = ?
```

Вывод

При выполнении этой лабораторной работы я столкнулся с проблемой: так как моя программа передавала в родительский процесс сообщения об ошибках то родительский процесс всегда ждал ,после передачи данных, обратную связь, которая при отсутствии ошибок не приходила. Поэтому потребовалось передавать сообщение “Success” при успешной записи в файл. В целом, эта лабораторная работа позволила хорошо разобраться в межпроцессорном взаимодействии.