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: Воронухин Н.А.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 10.12.24

Москва, 2024

Постановка задачи

Вариант 3.

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 pipe1, который связан с стандартным входным потоком дочернего процесса. Дочерний процесс при необходимости передает данные в родительский процесс через pipe2. 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 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); – создает pipe, однонаправленный канал связи между процессами.
- ssize_t write(int fd, const void buf[.count], size_t count); - пишет count байт из буфера buf в файл, на который ссылается файловый дескриптор fd.
- ssize_t read(int fd, void buf[.count], size_t count); - считывает count байт в буфер buf из файла, на который ссылается файловый дескриптор fd
- int open(const char *pathname, int flags, /* mode_t mode */); - открывает файл описанный в pathname, параметр флагов flags определяет способ открытия файла (только для чтения, записи и тд), аргумент mode задает права доступа к файлу, в случае создания нового файла.
- int close(int fd); - закрывает файловый дескриптор.
- 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.
- void exit(int status); - приводит к обычному завершению программы, и величина status возвращается процессу-родителю.
- int openat(int dirfd, const char *pathname, int flags, mode_t mode); - то же, что и open() но путь к фалуй относительный и в него передается дескриптор директории.
- exit_group – системный вызов завершающий все потоки процесса.
- int readlink(const char *path, char *buf, size_t bufsiz); - помещает содержимое символьной ссылки path в буфер buf длиной bufsiz.
- int dup2(int oldfd, int newfd); - dup2 делает файловый дескриптор newfd копией oldfd, закрывая newfd, если требуется.

Программа получает из стандартного потока ввода имя файла, открывает его, создает 2 канала pipe и создает дочерний процесс с помощью fork, загружая в него новую программу с помощью exec. Далее происходит разделение на 2 подпрограммы с помощью if по pid процесса, дочерний процесс посылает в pipe2 сообщение о готовности к чтению, родительский процесс считывает его, и начинает считывать данные из потока ввода, отправляя их через pipe1 дочернему процессу. Дочерний процесс анализирует ввод, производя, если необходимо вычисления. В случае ошибки, он отправляет через pipe2 сообщение об этом. Результаты вычисления дочерний процесс записывает в файл.

В случае получения делителя 0, который служит маркером для остановки, дочерний процесс отправляет через pipe2 сообщение об остановке родителю и завершается, записав последний результат в файл.

Родитель производит итерации в цикле, каждый раз сначала считывая сообщение полученное от дочернего процесса, о его состоянии, и дальше либо считывает последующие данные из потока ввода, либо завершается с ошибкой (в случае возникновения ошибки в дочернем процессе), либо завершается нормально, перед этим получив код завершения дочернего процесса через wait(NULL), чтобы избежать ситуации, когда дочерний процесс становится зомби.

Код программы

parent.c

```
#include <unistd.h>

#include <sys/types.h>

#include <fcntl.h>

#include <sys/stat.h>

#include <stdlib.h>

#include <ctype.h>

#include <sys/wait.h>

#include <stdio.h>

#include <unistd.h>



static const int FILE_NAME_BUF_SIZE = 15;

static const int STR_BUF_SIZE = 50;

static char const CHILDE_PROGRAM_NAME[] = "childe";

static const int PATH_SIZE = 1024;
```

```
    enum PIPE_SIGNALS {
        STOP_MES,
        READY_TO_READ,
        ERROR_MES,
    };

    enum ERR_FLG {
        // for parent.c and childe.c
        READ_ERR = 1,
        WRITE_ERR,
        WRONG_INSTRUCTION_ERR,
        // unic for parent.c
        WRONG_FILE_NAME_ERR,
        OPEN_FILE_ERR,
        CREATE_PIPE_ERR,
        CREATE_PROCESS_ERR,
        READ_PATH_ERR,
        CONVERT_ERR,
        CAT_ERR,
        EXEC_ERR,
        DUP2_ERR,
        CHILDE_ERROR,
    };


    void exitParent(int, char* const, int);


    int main() {

        char progpath[PATH_SIZE];
        ssize_t len = readlink("/proc/self/exe", progpath, sizeof(progpath) / sizeof(char)
- sizeof(char));
```

```
if (len == -1) {
    const char mes[] = "Error: failed to read full program path\n";
    exitParent(READ_PATH_ERR, mes, sizeof(mes));
}
while (progpath[len] != '/')
    --len;
progpath[len] = '\0';



char buff[FILE_NAME_BUF_SIZE];
int nameFileLen = 0;
if ((nameFileLen = read(STDIN_FILENO, buff, sizeof(buff))) == -1) {
    const char mes[] = "Error: failed to read from STDIN\n";
    exitParent(READ_ERR, mes, sizeof(mes));
}
else if (buff[nameFileLen - 1] != '\n') {
    const char mes[] = "Error: file name too long\n";
    exitParent(WRONG_FILE_NAME_ERR, mes, sizeof(mes));
}
buff[nameFileLen - 1] = '\0';



int fd = open(buff, O_WRONLY | O_CREAT | O_TRUNC, S_IRWXU | S_IRWXO);
if (fd == -1) {
    const char mes[] = "Error: failed to open file\n";
    exitParent(OPEN_FILE_ERR, mes, sizeof(mes));
}

int pToChilde[2];
if (pipe(pToChilde)) {
    close(fd);
    const char mes[] = "Error: failed to create pipe\n";
    exitParent(CREATE_PIPE_ERR, mes, sizeof(mes));
```

```
}
int pToParent[2];
if (pipe(pToParent)) {
    close(fd);
    close(pToChilde[1]);
    close(pToChilde[0]);
    const char mes[] = "Error: failed to create pipe\n";
    exitParent(CREATE_PIPE_ERR, mes, sizeof(mes));
}



pid_t pid = fork();
if (pid == -1) {
    close(fd);
    close(pToChilde[1]);
    close(pToChilde[0]);
    close(pToParent[1]);
    close(pToParent[0]);
    const char mes[] = "Error: failed to create process\n";
    exitParent(CREATE_PROCESS_ERR, mes, sizeof(mes));
}
else if (pid == 0) {
    close(pToChilde[1]);
    close(pToParent[0]);


    if (dup2(pToChilde[0], STDIN_FILENO) == -1){
        const char mes[] = "Error: failed to dup\n";
        exit(DUP2_ERR);
    }
    if (dup2(pToParent[1], STDOUT_FILENO) == -1){
        const char mes[] = "Error: failed to dup\n";
        exit(DUP2_ERR);
    }
```

```
            close(pToChilde[0]);
            close(pToParent[1]);

            char path[PATH_SIZE];
            if (snprintf(path, sizeof(path) - sizeof(char), "%s/%s", progpath,
CHILDE_PROGRAM_NAME) < 0){
                const char mes[] = "Error: failed to create full name of childe
process\n";
                write(STDERR_FILENO, mes, sizeof(mes));
                exit(CAT_ERR);
            }
            char fd_str[FILE_NAME_BUF_SIZE];
            if (snprintf(fd_str, sizeof(fd_str), "%d", fd) < 0) {
                const char mes[] = "Error: failed to convert int FD to str FD\n";
                write(STDERR_FILENO, mes, sizeof(mes));
                exit(CONVERT_ERR);
            }

            char* const argv[] = {CHILDE_PROGRAM_NAME, fd_str, NULL};

            int status = execv(path, argv);

            if (status == -1) {
                char mes[] = "Error: failed to exec into new exectuable image\n";
                write(STDERR_FILENO, mes, sizeof(mes));
                exit(EXEC_ERR);
            }
        }
        // parent
        else {
            close(pToChilde[0]);
            close(pToParent[1]);
            close(fd);
```

```
char strBuf[STR_BUF_SIZE];
char sigHand[1] = {(char)ERROR_MES};
while (1) {
    if (read(pToParent[0], sigHand, sizeof(sigHand)) == -1) {
        close(pToChilde[1]);
        close(pToParent[0]);
        wait(NULL);
        const char mes[] = "Error: failed to read from pipe\n";
        exitParent(READ_ERR, mes, sizeof(mes));
    }
    else if (sigHand[0] == (char)READY_TO_READ) {
        int number = read(STDIN_FILENO, strBuf, sizeof(strBuf));
        if (number == -1) {
            close(pToChilde[1]);
            close(pToParent[0]);
            wait(NULL);
            const char mes[] = "Error: failed to read from STDIN\n";
            exitParent(READ_ERR, mes, sizeof(mes));
        }
        else if (write(pToChilde[1], strBuf, number) == -1) {
            close(pToChilde[1]);
            close(pToParent[0]);
            wait(NULL);
            const char mes[] = "Error: failed to write to pipe\n";
            exitParent(WRITE_ERR, mes, sizeof(mes));
        }
    }
    else if (sigHand[0] == STOP_MES) {
        close(pToChilde[1]);
        close(pToParent[0]);
        wait(NULL);
```

```
                exit(EXIT_SUCCESS);
            }
            else {
                close(pToChilde[1]);
                close(pToParent[0]);
                const char mes[] = "Error: error in child process\n";
                int status = 0;
                wait(&status);
                exitParent(CHILDE_ERROR + status, mes, sizeof(mes));
            }
        }
    }
}

void exitParent(int code, char* const message, int mes_size) {
    write(STDERR_FILENO, message, mes_size);
    exit(code);
}
```

childe.c

```
#include <unistd.h>
#include <sys/types.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <ctype.h>
#include <sys/wait.h>
#include <stdio.h>
#include <string.h>

const int STR_BUF_SIZE = 50;
```

```
enum PIPE_SIGNALS {
    STOP_MES,
    READY_TO_READ,
    ERROR_MES,
};

typedef enum PARSER_FLAGS {
    CARRY_DIVIDED = 0b1,
    CARRY_DIVISOR = 0b10,
    SPACE_FOUND   = 0b100,
    ENDLINE_FOUND = 0b1000,
    CUR_VAL_DIGIT = 0b10000,
    CUR_VAL_SPACE = 0b100000,
    CUR_VAL_ENDL  = 0b1000000,
    CUR_VAL_SIGN  = 0b10000000,
    SIGN_FOUND    = 0b100000000,
    CUR_VAL_ZERO  = 0b1000000000,
    ZERO_FOUND    = 0b10000000000,
} PARSER_FLAGS;

typedef enum ERR_FLG {
    READ_ERR = 1,
    WRITE_ERR,
    WRONG_INSTRUCTION_ERR,
} ERR_FLG;


void exitChilde(int, int, int, char*, int);
void stopChilde(float, int, int);


int main(int argc,char **argv) {
```

```
            int fd = atoi(argv[1]);
            char strBuf[STR_BUF_SIZE];
            float divided = 0;
            int divisor = 0;
            int parser_flag = CARRY_DIVIDED | ENDLINE_FOUND;
            while (1) {
                int number = 0;
                char sig[1] = {(char)READY_TO_READ};
                if (write(STDOUT_FILENO, sig, sizeof(sig)) == -1) {
                    const char mes[] = "Error: failed to write to pipe\n";
                    exitChilde(WRITE_ERR, fd, ERROR_MES, mes, sizeof(mes));
                }
                else if ((number = read(STDIN_FILENO, strBuf, (int)sizeof(strBuf))) == -1)
{
                    const char mes[] = "Error: failed to read from pipe\n";
                    exitChilde(READ_ERR, fd, ERROR_MES, mes, sizeof(mes));
                }
                number /= sizeof(char);
                char* ptr = strBuf;
                for (; ptr < strBuf + number; ++ptr) {
                    if (*ptr == '0' && (parser_flag & (ENDLINE_FOUND | SPACE_FOUND |
SIGN_FOUND | ZERO_FOUND)))
                        parser_flag |= CUR_VAL_ZERO;
                    else if (isdigit(*ptr)) parser_flag |= CUR_VAL_DIGIT;
                    else if (*ptr == ' ') parser_flag |= CUR_VAL_SPACE;
                    else if (*ptr == '\n') parser_flag |= CUR_VAL_ENDL;
                    else if (*ptr == '-' || *ptr == '+') parser_flag |= CUR_VAL_SIGN;

                    switch (parser_flag) {

                        case CARRY_DIVIDED | CUR_VAL_DIGIT:
                        case CARRY_DIVIDED | CUR_VAL_DIGIT | ENDLINE_FOUND:
                            divided = divided * 10 + *ptr - '0';
                            parser_flag = CARRY_DIVIDED;
```

```
    break;
case CARRY_DIVIDED | CUR_VAL_DIGIT | SIGN_FOUND:
    divided = divided * (*ptr - '0');
    parser_flag = CARRY_DIVIDED;
    break;


case CARRY_DIVIDED | CUR_VAL_SPACE:
case CARRY_DIVIDED | CUR_VAL_SPACE | ZERO_FOUND:
    parser_flag = CARRY_DIVISOR | SPACE_FOUND;
    break;


case CARRY_DIVIDED | CUR_VAL_ENDL:
case CARRY_DIVIDED | CUR_VAL_ENDL | ZERO_FOUND:
    char* result[14];
    int len = sprintf(result, "%g\n", divided);
    if (write(fd, result, len * sizeof(char)) == -1) {
        const char mes[] = "Error: failed to write to file\n";
        exitChilde(WRITE_ERR, fd, ERROR_MES, mes, sizeof(mes));
    }
    divided = 0;
    parser_flag = CARRY_DIVIDED | ENDLINE_FOUND;
    break;


case CARRY_DIVIDED | CUR_VAL_SIGN | ENDLINE_FOUND:
    if (*ptr == '-') divided = -1;
    else divided = 1;
    parser_flag = CARRY_DIVIDED | SIGN_FOUND;
    break;
```

```
case CARRY_DIVIDED | CUR_VAL_ZERO | ENDLINE_FOUND:
case CARRY_DIVIDED | CUR_VAL_ZERO | SIGN_FOUND:
    divided = 0;
    parser_flag = CARRY_DIVIDED | ZERO_FOUND;
    break;



case CARRY_DIVISOR | CUR_VAL_DIGIT:
case CARRY_DIVISOR | CUR_VAL_DIGIT | SPACE_FOUND:
    divisor = divisor * 10 + *ptr - '0';
    parser_flag = CARRY_DIVISOR;
    break;
case CARRY_DIVISOR | CUR_VAL_DIGIT | SIGN_FOUND:
    divisor = divisor * (*ptr - '0');
    parser_flag = CARRY_DIVISOR;
    break;



case CARRY_DIVISOR | CUR_VAL_SPACE:
case CARRY_DIVISOR | CUR_VAL_SPACE | ZERO_FOUND:
    if (divisor == 0) {
        stopChilde(divided, fd, STOP_MES);
        divided = 0;
    }
    else {
        divided /= divisor;
        divisor = 0;
        parser_flag = CARRY_DIVISOR | SPACE_FOUND;
    }
    break;
```

```
                        case CARRY_DIVISOR | CUR_VAL_ENDL:
                        case CARRY_DIVISOR | CUR_VAL_ENDL | ZERO_FOUND:
                            if (divisor == 0) {
                                stopChilde(divided, fd, STOP_MES);
                                divided = 0;
                            }
                            else {
                                divided /= divisor;
                                divisor = 0;
                                char* result[14];
                                int len = sprintf(result, "%g\n", divided);
                                if (write(fd, result, sizeof(char) * len) == -1) {
                                    const char mes[] = "Error: failed to write to file\n";
                                    exitChilde(WRITE_ERR, fd, ERROR_MES, mes,
sizeof(mes));
                                }
                                divided = 0;
                                parser_flag = CARRY_DIVIDED | ENDLINE_FOUND;
                            }
                            break;


                        case CARRY_DIVISOR | CUR_VAL_SIGN | SPACE_FOUND:
                            if (*ptr == '-') divisor = -1;
                            else divisor = 1;
                            parser_flag = CARRY_DIVISOR | SIGN_FOUND;
                            break;


                        case CARRY_DIVISOR | CUR_VAL_ZERO | SPACE_FOUND:
                        case CARRY_DIVISOR | CUR_VAL_ZERO | SIGN_FOUND:
                            divisor = 0;
                            parser_flag = CARRY_DIVISOR | ZERO_FOUND;
```

```
                            break;


                        default:
                        /*
                            ERRORS:
                            case CARRY_DIVIDED | CUR_VAL_SPACE | ENDLINE_FOUND
                            case CARRY_DIVIDED | CUR_VAL_ENDL | ENDLINE_FOUND
                            case CARRY_DIVISOR | CUR_VAL_SPACE | SPACE_FOUND
                            case CARRY_DIVISOR | CUR_VAL_ENDL | SPACE_FOUND
                            case CARRY_DIVIDED | CUR_VAL_SIGN ...
                            IMPOSSIBLE:
                            case CARRY_DIVIDED | CUR_VAL_DIGIT | SPACE_FOUND : SPACE_FOUND
-> CARRY_DIVISOR
                            case CARRY_DIVIDED | CUR_VAL_SPACE | SPACE_FOUND
                            case CARRY_DIVIDED | CUR_VAL_ENDL | SPACE_FOUND
                            case CARRY_DIVISOR | CUR_VAL_DIGIT | ENDLINE_FOUND :
ENDLINE_FOUND -> CARRY_DIVIDED
                            case CARRY_DIVISOR | CUR_VAL_SPACE | ENDLINE_FOUND
                            case CARRY_DIVISOR | CUR_VAL_SPACE | ENDLINE_FOUND
                            case CARRY_DIVIDED | CUR_VAL_SIGN | SPACE_FOUND ...
                        */
                            const char mes[] = "Error: wrong inscturction\n";
                            exitChilde(WRONG_INSTRUCTION_ERR, fd, ERROR_MES, mes,
sizeof(mes));
                            break;
                    }
                }
            }
    }


    void exitChilde(int code, int fd, int sig, char* const mes, int mes_size) {
        char buf[1] = {(char)sig};
        write(STDOUT_FILENO, buf, sizeof(buf));
```

```
    write(STDERR_FILENO, mes, mes_size);
    close(fd);
    exit(code);
}

void stopChilde(float divided, int fd, int mes_to_parent) {
    char res[14];
    int len = sprintf(res, "%g\n", divided);
    if (write(fd, res, sizeof(char) * len) == -1) {
        const char mes[] = "Error: failed to write to file\n";
        exitChilde(WRITE_ERR, fd, ERROR_MES, mes, sizeof(mes));
    }
    res[0] = (char)mes_to_parent;
    if (write(STDOUT_FILENO, res, sizeof(char)) == -1) {
        const char mes[] = "Error: failed to write to pipe\n";
        exitChilde(WRITE_ERR, fd, ERROR_MES, mes, sizeof(mes));
    }
    close(fd);
    exit(EXIT_SUCCESS);
}
```

Протокол работы программы

Тестирование:

```
vnadez@dbnicknv:~/Projects/OS_LABS/LAB_1$txt.txt
10 10 10
12 334 -56 43 5654 3 3
123 324 3345
-234 2
32
-3
2
0
```

```
4 0

------------

in txt1.txt:

0.1

-3.73177e-10

0.000113492

-83

32

-3

2

0

vnadez@dbnicknv:~/Projects/OS_LABS/LAB_1$txt2.txt

67

-7 -7 -7 -7 -7 -7 -7 -7 -7 -7

0 0

--------

in txt2.txt:

67

1.73467e-07

0
```

Strace:

Ввод:

```
vnadez@dbnicknv:~/Projects/OS_LABS/LAB_1$ strace -f --output=outStrace.txt ./parent

txt.txt

12 12 12 -1

7823 238 1298 1287 1287 1289 7621 1672 1276 1267 1286 2167 1267 1276 2167 21678

0 1 2 2 3 4

45 54 45

12 0
```

Вывод:

```
25128 execve("./parent", ["./parent"], 0x7ffcecaf6a80 /* 64 vars */) = 0

25128 brk(NULL)                         = 0x56388599c000

25128 mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f5bef5d2000
```

```
     25128 access("/etc/ld.so.preload", R_OK) = -1 ENOENT (Нет такого файла или каталога)

     25128 openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

     25128 newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=92170, ...}, AT_EMPTY_PATH) = 0

     25128 mmap(NULL, 92170, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f5bef5bb000

     25128 close(3)                          = 0

     25128 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3

     25128 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

     25128 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

     25128 newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=1922136, ...}, AT_EMPTY_PATH) =
0

     25128 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

     25128 mmap(NULL, 1970000, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f5bef3da000

     25128 mmap(0x7f5bef400000, 1396736, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x26000) = 0x7f5bef400000

     25128 mmap(0x7f5bef555000, 339968, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x17b000) = 0x7f5bef555000

     25128 mmap(0x7f5bef5a8000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1ce000) = 0x7f5bef5a8000

     25128 mmap(0x7f5bef5ae000, 53072, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f5bef5ae000

     25128 close(3)                          = 0

     25128 mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f5bef3d7000

     25128 arch_prctl(ARCH_SET_FS, 0x7f5bef3d7740) = 0

     25128 set_tid_address(0x7f5bef3d7a10)   = 25128

     25128 set_robust_list(0x7f5bef3d7a20, 24) = 0

     25128 rseq(0x7f5bef3d8060, 0x20, 0, 0x53053053) = 0

     25128 mprotect(0x7f5bef5a8000, 16384, PROT_READ) = 0

     25128 mprotect(0x563883b66000, 4096, PROT_READ) = 0

     25128 mprotect(0x7f5bef604000, 8192, PROT_READ) = 0

     25128 prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY})
= 0

     25128 munmap(0x7f5bef5bb000, 92170)     = 0

     25128 readlink("/proc/self/exe", "/home/vnadez/Projects/OS_LABS/LA"..., 1023) = 42

     25128 read(0, "txt.txt\n", 15)          = 8

     25128 openat(AT_FDCWD, "txt.txt", O_WRONLY|O_CREAT|O_TRUNC, 0707) = 3

     25128 pipe2([4, 5], 0)                  = 0
```

```
     25128 pipe2([6, 7], 0)                  = 0

     25128 clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7f5bef3d7a10) = 25194

     25194 set_robust_list(0x7f5bef3d7a20, 24 <unfinished ...>

     25128 close(4 <unfinished ...>

     25194 <... set_robust_list resumed>)    = 0

     25128 <... close resumed>)              = 0

     25194 close(5 <unfinished ...>

     25128 close(7 <unfinished ...>

     25194 <... close resumed>)              = 0

     25128 <... close resumed>)              = 0

     25194 close(6 <unfinished ...>

     25128 close(3 <unfinished ...>

     25194 <... close resumed>)              = 0

     25128 <... close resumed>)              = 0

     25128 read(6,  <unfinished ...>

     25194 dup2(4, 0)                        = 0

     25194 dup2(7, 1)                        = 1

     25194 close(4)                          = 0

     25194 close(7)                          = 0

     25194 execve("/home/vnadez/Projects/OS_LABS/LAB_1/childe", ["childe", "3"],
0x7fffa5c4a098 /* 64 vars */) = 0

     25194 brk(NULL)                         = 0x55ce44b0f000

     25194 mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f6c4c3a9000

     25194 access("/etc/ld.so.preload", R_OK) = -1 ENOENT (Нет такого файла или каталога)

     25194 openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 4

     25194 newfstatat(4, "", {st_mode=S_IFREG|0644, st_size=92170, ...}, AT_EMPTY_PATH) = 0

     25194 mmap(NULL, 92170, PROT_READ, MAP_PRIVATE, 4, 0) = 0x7f6c4c392000

     25194 close(4)                          = 0

     25194 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 4

     25194 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

     25194 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

     25194 newfstatat(4, "", {st_mode=S_IFREG|0755, st_size=1922136, ...}, AT_EMPTY_PATH) =
0

     25194 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
```

```
     25194 mmap(NULL, 1970000, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 4, 0) = 0x7f6c4c1b1000

     25194 mmap(0x7f6c4c1d7000, 1396736, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 4, 0x26000) = 0x7f6c4c1d7000

     25194 mmap(0x7f6c4c32c000, 339968, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 4,
0x17b000) = 0x7f6c4c32c000

     25194 mmap(0x7f6c4c37f000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 4, 0x1ce000) = 0x7f6c4c37f000

     25194 mmap(0x7f6c4c385000, 53072, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f6c4c385000

     25194 close(4)                          = 0

     25194 mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f6c4c1ae000

     25194 arch_prctl(ARCH_SET_FS, 0x7f6c4c1ae740) = 0

     25194 set_tid_address(0x7f6c4c1aea10)   = 25194

     25194 set_robust_list(0x7f6c4c1aea20, 24) = 0

     25194 rseq(0x7f6c4c1af060, 0x20, 0, 0x53053053) = 0

     25194 mprotect(0x7f6c4c37f000, 16384, PROT_READ) = 0

     25194 mprotect(0x55ce43363000, 4096, PROT_READ) = 0

     25194 mprotect(0x7f6c4c3db000, 8192, PROT_READ) = 0

     25194 prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY})
= 0

     25194 munmap(0x7f6c4c392000, 92170)     = 0

     25194 write(1, "\1", 1)                 = 1

     25128 <... read resumed>"\1", 1)        = 1

     25194 read(0,  <unfinished ...>

     25128 read(0, "12 12 12 -1\n", 50)      = 12

     25128 write(5, "12 12 12 -1\n", 12)     = 12

     25194 <... read resumed>"12 12 12 -1\n", 50) = 12

     25128 read(6,  <unfinished ...>

     25194 write(3, "-0.0833333\n", 11)      = 11

     25194 write(1, "\1", 1)                 = 1

     25128 <... read resumed>"\1", 1)        = 1

     25194 read(0,  <unfinished ...>

     25128 read(0, "7823 238 1298 1287 1287 1289 762"..., 50) = 50

     25128 write(5, "7823 238 1298 1287 1287 1289 762"..., 50) = 50

     25194 <... read resumed>"7823 238 1298 1287 1287 1289 762"..., 50) = 50

     25128 read(6,  <unfinished ...>

     25194 write(1, "\1", 1 <unfinished ...>
```

```
25128 <... read resumed>"\1", 1)        = 1
25194 <... write resumed>)              = 1
25128 read(0,  <unfinished ...>
25194 read(0,  <unfinished ...>
25128 <... read resumed>"286 2167 1267 1276 2167 21678\n", 50) = 30
25128 write(5, "286 2167 1267 1276 2167 21678\n", 30) = 30
25194 <... read resumed>"286 2167 1267 1276 2167 21678\n", 50) = 30
25128 read(6,  <unfinished ...>
25194 write(3, "2.8026e-45\n", 11)      = 11
25194 write(1, "\1", 1)                 = 1
25128 <... read resumed>"\1", 1)        = 1
25194 read(0,  <unfinished ...>
25128 read(0, "0 1 2 2 3 4\n", 50)      = 12
25128 write(5, "0 1 2 2 3 4\n", 12)     = 12
25194 <... read resumed>"0 1 2 2 3 4\n", 50) = 12
25128 read(6,  <unfinished ...>
25194 write(3, "0\n", 2)                = 2
25194 write(1, "\1", 1)                 = 1
25128 <... read resumed>"\1", 1)        = 1
25194 read(0,  <unfinished ...>
25128 read(0, "45 54 45\n", 50)         = 9
25128 write(5, "45 54 45\n", 9)         = 9
25194 <... read resumed>"45 54 45\n", 50) = 9
25128 read(6,  <unfinished ...>
25194 write(3, "0.0185185\n", 10)       = 10
25194 write(1, "\1", 1)                 = 1
25128 <... read resumed>"\1", 1)        = 1
25194 read(0,  <unfinished ...>
25128 read(0, "12 0\n", 50)             = 5
25128 write(5, "12 0\n", 5)             = 5
25194 <... read resumed>"12 0\n", 50)   = 5
25128 read(6,  <unfinished ...>
25194 write(3, "12\n", 3)                = 3
25194 write(1, "\0", 1)                 = 1
25128 <... read resumed>"\0", 1)        = 1
```

```
25128 close(5 <unfinished ...>
25194 close(3 <unfinished ...>
25128 <... close resumed>)              = 0
25194 <... close resumed>)              = 0
25128 close(6)                          = 0
25194 exit_group(0)                     = ?
25128 wait4(-1,  <unfinished ...>
25194 +++ exited with 0 +++
25128 <... wait4 resumed>NULL, 0, NULL) = 25194
25128 --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=25194, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
25128 exit_group(0)                     = ?
25128 +++ exited with 0 +++
```

Вывод

При выполнении этой работы я получил практические навыки в управлении процессами посредством системных вызовов, и в организации межпроцессорного взаимодействия посредством каналов.