Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

**Лабораторная работа №3 по курсу**

**«Операционные системы»**

Группа: М8О-206Б-22

Студент: Ларин И.А.

Преподаватель: Миронов Е.С.

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

Дата: 14.11.2023

Москва, 2023

**Постановка задачи**

**Группа вариантов 4.**

Родительский процесс создает дочерний процесс. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child проверяет строки на валидность правилу. Если строка соответствует правилу, то она выводится в стандартный поток вывода дочернего процесса, иначе в pipe2 выводится информация об ошибке. Родительский процесс полученные от child ошибки выводит в стандартный поток вывода.

**Вариант 15.**

Правило проверки: строка должна начинаться с заглавной буквы.

**Общий метод и алгоритм решения**

Использованные системные вызовы:

* pid\_t **fork**(void); – создает дочерний процесс.
* **execl**(“path\_to\_file”, “path\_to\_file”, args, … , NULL) - запускает программу на выполнение.
* **dup2**(int id1, int id2) - вызов позволяет сделать так, чтобы то, что использовалось в качестве id1 теперь будет использоваться в качестве id2.
* void\* **mmap**(void \*addr, size\_t length, int prot, int flags, int fd, off\_t offset) создает новое отображение в виртуальном адресном пространстве вызывающего процесса. Начальный адрес для нового сопоставления указан в addr. Аргумент length указывает длину сопоставления (которая должна быть больше 0), fd - файловый дескриптор, flags - атрибуты, offset - смещение относительно начала области, которое должно быть кратно размеру страницы, возвращаемому sysconf(\_SC\_PAGE\_SIZE).
* int **munmap**(void \*addr, size\_t length) - системный вызов удаляет отображения для указанного диапазона адресов и приводит к тому, что дальнейшие ссылки на адреса в пределах диапазона генерируют недопустимые ссылки на память. Регион также автоматически размечается при завершении процесса. С другой стороны, закрытие файлового дескриптора не отменяет отображение региона.

В начале в main мы вводим название файла, потом создаём три файловых дескриптора, обрезаем до определённого размера, получаем идентификатор родительского процесса, перенаправляем ответ файл на вывод, вызываем дочерний процесс, в родительском вызываем mmap для file\_message и для file\_errors и закрываем для него file\_answers. Функция loop считывает строчку и отправляет на проверку дочернему процессу. Дочерний процесс в это время открывает file\_message и file\_errors, вызывает mmap для file\_messages и file\_errors, принимает сигнал от родительского процесса, выполняет проверку правильности строки, если строка с маленькой буквы, то отправляет сообщение родительскому процессу, который выводит о неправильности строки, если строка с большой буквы, то она записывается в файл. Вся программа выполняется пока не будет введён символ обозначающий конец файла.

**Код программы**

Main.c

#include <stdio.h>

#include <unistd.h>

#include <sys/wait.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <string.h>

#include <fcntl.h>

#include <errno.h>

#include "d\_string.h"

#include "sync\_constants.h"

#include <signal.h>

char\* ERROR\_PTR;

void accept\_err(int signum){

    write(STDOUT\_FILENO, ERROR\_PTR, sizeof(char)\*strlen(ERROR\_PTR));

}

int loop(char\* msg\_ptr, pid\_t child\_pid) {

    errno = 0;

    char\* st = NULL;

    int st\_len = 0;

    int scan\_res;

    do {

        scan\_res = scan\_string(&st, &st\_len);

        st[st\_len] = (scan\_res == EOF)? EOF : '\n';

        for (int i=0; i <= st\_len; ++i) {

        msg\_ptr[i] = st[i];

        }

        kill(child\_pid, CHILD\_SIGNAL\_CHECK);

    } while (scan\_res != EOF);

    return errno;

}

int main() {

    errno = 0;

    char\* filename = NULL;

    int filename\_len = 0;

    int file\_answers, file\_messages, file\_errors;

    scan\_string(&filename, &filename\_len);

    file\_answers = creat(filename, S\_IRWXU);

    file\_messages = open(MESSAGES\_FILENAME, O\_RDWR|O\_CREAT, S\_IRWXU);

    file\_errors = open(ERRORS\_FILENAME, O\_RDWR|O\_CREAT, S\_IRWXU);

    ftruncate(file\_messages, MESSAGES\_FILESIZE);

    ftruncate(file\_errors, ERRORS\_FILESIZE);

    int sync\_st = getpid();

    write(file\_messages, &sync\_st, sizeof(sync\_st));

    pid\_t id = fork();

    if (id < 0)

        return errno;

    else if (id == 0) {

        dup2(file\_answers, STDOUT\_FILENO);

        execl("./child", "./child", (char \*) NULL);

    }

    signal(PARENT\_SIGNAL\_CHECK, accept\_err);

    char \*msg\_ptr = mmap(NULL, MESSAGES\_FILESIZE\*sizeof(char),PROT\_WRITE, MAP\_SHARED, file\_messages, 0);

    if (errno)

        return errno;

    ERROR\_PTR = mmap(NULL, ERRORS\_FILESIZE\*sizeof(char),PROT\_READ, MAP\_SHARED, file\_errors, 0);

    if (errno)

        return errno;

    else if (loop(msg\_ptr, id))

        return errno;

    else if (munmap(msg\_ptr, MESSAGES\_FILESIZE\*sizeof(char)) == -1)

        return errno;

    else if (munmap(ERROR\_PTR, ERRORS\_FILESIZE\*sizeof(char)) == -1)

        return errno;

    int status;

    waitpid(id, &status, 0);

    return status;

}

Child.c

#include <stdio.h>

#include <unistd.h>

#include <string.h>

#include <errno.h>

#include <fcntl.h>

#include <sys/wait.h>

#include <sys/mman.h>

#include <stdbool.h>

#include <malloc.h>

#include <stdlib.h>

#include "sync\_constants.h"

int SYNC\_ST;

char\* MESSAGE\_PTR;

char\* ERROR\_PTR;

int LEN\_READ\_ST = 0;

int MAX\_READ\_LEN = 1;

char\* READ\_ST = NULL;

void read\_chunk(int signum) {

    if (READ\_ST == NULL)

        READ\_ST = malloc(MAX\_READ\_LEN\*sizeof(char));

    int i=0;

    for (; MESSAGE\_PTR[i] != '\n' && MESSAGE\_PTR[i] != EOF; ++i) {

        READ\_ST[LEN\_READ\_ST++] = MESSAGE\_PTR[i];

        if (LEN\_READ\_ST >= MAX\_READ\_LEN) {

            MAX\_READ\_LEN \*= 2;

            READ\_ST = realloc(READ\_ST, MAX\_READ\_LEN\*sizeof(char));

        }

    }

    char last = MESSAGE\_PTR[i];

    if ((last == '\n' || last == EOF) && i != 0) {

        if ('A' <= READ\_ST[0] && READ\_ST[0] <= 'Z') {

            write(STDOUT\_FILENO, READ\_ST, LEN\_READ\_ST \* sizeof(char));

        } else {

            strcpy(ERROR\_PTR, "invalid string\n");

            kill(SYNC\_ST, PARENT\_SIGNAL\_CHECK);

        }

        LEN\_READ\_ST = 0;

    }

    if (last == EOF)

        exit(0);

}

int main() {

    errno = 0;

    int file\_messages, file\_errors;

    file\_messages = open(MESSAGES\_FILENAME, O\_RDWR);

    file\_errors = open(ERRORS\_FILENAME, O\_RDWR);

    read(file\_messages, &SYNC\_ST, sizeof(SYNC\_ST));

    MESSAGE\_PTR = mmap(NULL, MESSAGES\_FILESIZE\*sizeof(char),PROT\_READ, MAP\_SHARED, file\_messages, 0);

    if (errno)

        return errno;

    ERROR\_PTR = mmap(NULL, ERRORS\_FILESIZE\*sizeof(char), PROT\_WRITE, MAP\_SHARED, file\_errors, 0);

    if (errno)

        return errno;

    signal(CHILD\_SIGNAL\_CHECK, read\_chunk);

    while (true);

}

D\_string.c

#include "d\_string.h"

#include <stdlib.h>

#include <stdio.h>

#include <errno.h>

#include <unistd.h>

int scan\_string(char\*\* str, int\* len) {

    int buf\_size = 1;

    int new\_len = 0;

    char \*new\_str = malloc(buf\_size\*sizeof(char));

    if (new\_str == NULL)

        return -1;

    int c;

    while ((c = getchar()) > 0 && c != '\n') {

        new\_str[new\_len++] = (char)c;

        if (new\_len >= buf\_size) {

            buf\_size \*= 2;

            new\_str = realloc(new\_str, buf\_size\*sizeof(char));

            if (new\_str == NULL)

                return -1;

        }

    }

    new\_str[new\_len] = '\0';

    free(\*str);

    \*str = new\_str;

    \*len = new\_len;

    if (c == EOF)

        return EOF;

    return 0;

}

Sync\_constants.h

#include <signal.h>

const char MESSAGES\_FILENAME[] = "messages";

const char ERRORS\_FILENAME[] = "errors";

const int CHILD\_SIGNAL\_CHECK = SIGUSR1;

const int PARENT\_SIGNAL\_CHECK = SIGUSR1;

const int MESSAGES\_FILESIZE = 1024;

const int ERRORS\_FILESIZE = 128;

**Протокол работы программы**

**Тестирование:**

ilya@ilya-ilyal:~/osi3$ ./main

file.txt

fsd

invalid string

fdfd

invalid string

dsfsdf

invalid string

HNJK

HJK

HBJ

JMK

NJMK

JNMK

ds

**Strace:**

ilya@ilya-ilyal:~/osi3$ strace -f ./main

execve("./main", ["./main"], 0x7ffcc1fccc78 /\* 59 vars \*/) = 0

brk(NULL) = 0x560832cbc000

arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7ffc29dc4eb0) = -1 EINVAL (Недопустимый аргумент)

**mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f70da7ea000**

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

newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=53247, ...}, AT\_EMPTY\_PATH) = 0

**mmap(NULL, 53247, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f70da7dd000**

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\0\244;\374\204(\337f#\315I\214\234\f\256\271\32"..., 68, 896) = 68

newfstatat(3, "", {st\_mode=S\_IFREG|0755, st\_size=2216304, ...}, 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, 2260560, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f70da400000**

**mmap(0x7f70da428000, 1658880, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x7f70da428000**

**mmap(0x7f70da5bd000, 360448, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1bd000) = 0x7f70da5bd000**

**mmap(0x7f70da615000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x214000) = 0x7f70da615000**

**mmap(0x7f70da61b000, 52816, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f70da61b000**

**close(3) = 0**

**mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f70da7da000**

arch\_prctl(ARCH\_SET\_FS, 0x7f70da7da740) = 0

set\_tid\_address(0x7f70da7daa10) = 5525

set\_robust\_list(0x7f70da7daa20, 24) = 0

rseq(0x7f70da7db0e0, 0x20, 0, 0x53053053) = 0

mprotect(0x7f70da615000, 16384, PROT\_READ) = 0

mprotect(0x5608328d0000, 4096, PROT\_READ) = 0

mprotect(0x7f70da824000, 8192, PROT\_READ) = 0

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

munmap(0x7f70da7dd000, 53247) = 0

getrandom("\x8c\xe1\xf5\x35\xdc\x41\xcd\xa8", 8, GRND\_NONBLOCK) = 8

brk(NULL) = 0x560832cbc000

brk(0x560832cdd000) = 0x560832cdd000

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

read(0, file.txt

"file.txt\n", 1024) = 9

creat("file.txt", 0700) = 3

openat(AT\_FDCWD, "messages", O\_RDWR|O\_CREAT, 0700) = 4

openat(AT\_FDCWD, "errors", O\_RDWR|O\_CREAT, 0700) = 5

ftruncate(4, 1024) = 0

ftruncate(5, 128) = 0

getpid() = 5525

write(4, "\225\25\0\0", 4) = 4

clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLDstrace: Process 5586 attached

, child\_tidptr=0x7f70da7daa10) = 5586

[pid 5586] set\_robust\_list(0x7f70da7daa20, 24 <unfinished ...>

[pid 5525] rt\_sigaction(SIGUSR1, {sa\_handler=0x5608328ce3a9, sa\_mask=[USR1], sa\_flags=SA\_RESTORER|SA\_RESTART, sa\_restorer=0x7f70da442520}, <unfinished ...>

[pid 5586] <... set\_robust\_list resumed>) = 0

[pid 5525] <... rt\_sigaction resumed>{sa\_handler=SIG\_DFL, sa\_mask=[], sa\_flags=0}, 8) = 0

[pid 5525] **mmap(NULL, 1024, PROT\_WRITE, MAP\_SHARED, 4, 0 <unfinished ...>**

[pid 5586] dup2(3, 1 <unfinished ...>

[pid 5525] <... mmap resumed>) = 0x7f70da823000

[pid 5586] <... dup2 resumed>) = 1

[pid 5525] **mmap(NULL, 128, PROT\_READ, MAP\_SHARED, 5, 0 <unfinished** ...>

[pid 5586] execve("./child", ["./child"], 0x7ffc29dc5088 /\* 59 vars \*/ <unfinished ...>

[pid 5525] <... mmap resumed>) = 0x7f70da7e9000

[pid 5525] read(0, <unfinished ...>

[pid 5586] <... execve resumed>) = 0

[pid 5586] brk(NULL) = 0x55f594636000

[pid 5586] arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7ffcca0a34b0) = -1 EINVAL (Недопустимый аргумент)

[pid 5586] **mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7ff3ac83b000**

[pid 5586] access("/etc/ld.so.preload", R\_OK) = -1 ENOENT (Нет такого файла или каталога)

[pid 5586] openat(AT\_FDCWD, "/etc/ld.so.cache", O\_RDONLY|O\_CLOEXEC) = 6

[pid 5586] newfstatat(6, "", {st\_mode=S\_IFREG|0644, st\_size=53247, ...}, AT\_EMPTY\_PATH) = 0

[pid 5586] **mmap(NULL, 53247, PROT\_READ, MAP\_PRIVATE, 6, 0) = 0x7ff3ac82e000**

[pid 5586] close(6) = 0

[pid 5586] openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libc.so.6", O\_RDONLY|O\_CLOEXEC) = 6

[pid 5586] read(6, "\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

[pid 5586] pread64(6, "\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 5586] pread64(6, "\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

[pid 5586] pread64(6, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\244;\374\204(\337f#\315I\214\234\f\256\271\32"..., 68, 896) = 68

[pid 5586] newfstatat(6, "", {st\_mode=S\_IFREG|0755, st\_size=2216304, ...}, AT\_EMPTY\_PATH) = 0

[pid 5586] pread64(6, "\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 5586] **mmap(NULL, 2260560, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 6, 0) = 0x7ff3ac600000**

[pid 5586] **mmap(0x7ff3ac628000, 1658880, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 6, 0x28000) = 0x7ff3ac628000**

[pid 5586] **mmap(0x7ff3ac7bd000, 360448, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 6, 0x1bd000) = 0x7ff3ac7bd000**

[pid 5586] **mmap(0x7ff3ac815000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 6, 0x214000) = 0x7ff3ac815000**

[pid 5586] **mmap(0x7ff3ac81b000, 52816, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7ff3ac81b000**

[pid 5586] close(6) = 0

[pid 5586] **mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7ff3ac82b000**

[pid 5586] arch\_prctl(ARCH\_SET\_FS, 0x7ff3ac82b740) = 0

[pid 5586] set\_tid\_address(0x7ff3ac82ba10) = 5586

[pid 5586] set\_robust\_list(0x7ff3ac82ba20, 24) = 0

[pid 5586] rseq(0x7ff3ac82c0e0, 0x20, 0, 0x53053053) = 0

[pid 5586] mprotect(0x7ff3ac815000, 16384, PROT\_READ) = 0

[pid 5586] mprotect(0x55f593e3b000, 4096, PROT\_READ) = 0

[pid 5586] mprotect(0x7ff3ac875000, 8192, PROT\_READ) = 0

[pid 5586] prlimit64(0, RLIMIT\_STACK, NULL, {rlim\_cur=8192\*1024, rlim\_max=RLIM64\_INFINITY}) = 0

[pid 5586] munmap(0x7ff3ac82e000, 53247) = 0

[pid 5586] openat(AT\_FDCWD, "messages", O\_RDWR) = 6

[pid 5586] openat(AT\_FDCWD, "errors", O\_RDWR) = 7

[pid 5586] read(6, "\225\25\0\0", 4) = 4

[pid 5586] **mmap(NULL, 1024, PROT\_READ, MAP\_SHARED, 6, 0) = 0x7ff3ac874000**

[pid 5586] **mmap(NULL, 128, PROT\_WRITE, MAP\_SHARED, 7, 0) = 0x7ff3ac83a000**

[pid 5586] rt\_sigaction(SIGUSR1, {sa\_handler=0x55f593e392a9, sa\_mask=[USR1], sa\_flags=SA\_RESTORER|SA\_RESTART, sa\_restorer=0x7ff3ac642520}, {sa\_handler=SIG\_DFL, sa\_mask=[], sa\_flags=0}, 8) = 0

small

[pid 5525] <... read resumed>"small\n", 1024) = 6

[pid 5525] kill(5586, SIGUSR1) = 0

[pid 5586] --- SIGUSR1 {si\_signo=SIGUSR1, si\_code=SI\_USER, si\_pid=5525, si\_uid=1000} ---

[pid 5525] read(0, <unfinished ...>

[pid 5586] getrandom("\x60\xd5\x87\xd0\x49\x8d\xaf\xda", 8, GRND\_NONBLOCK) = 8

[pid 5586] brk(NULL) = 0x55f594636000

[pid 5586] brk(0x55f594657000) = 0x55f594657000

[pid 5586] kill(5525, SIGUSR1 <unfinished ...>

[pid 5525] <... read resumed>0x560832cbc2c0, 1024) = ? ERESTARTSYS (To be restarted if SA\_RESTART is set)

[pid 5586] <... kill resumed>) = 0

[pid 5525] --- SIGUSR1 {si\_signo=SIGUSR1, si\_code=SI\_USER, si\_pid=5586, si\_uid=1000} ---

[pid 5525] write(1, "invalid string\n", 15invalid string

) = 15

[pid 5525] rt\_sigreturn({mask=[]} <unfinished ...>

[pid 5586] rt\_sigreturn({mask=[]} <unfinished ...>

[pid 5525] <... rt\_sigreturn resumed>) = 0

[pid 5525] read(0, <unfinished ...>

[pid 5586] <... rt\_sigreturn resumed>) = 0

Big

[pid 5525] <... read resumed>"Big\n", 1024) = 4

[pid 5525] kill(5586, SIGUSR1) = 0

[pid 5586] --- SIGUSR1 {si\_signo=SIGUSR1, si\_code=SI\_USER, si\_pid=5525, si\_uid=1000} ---

[pid 5525] read(0, <unfinished ...>

[pid 5586] write(1, "Big", 3) = 3

[pid 5586] rt\_sigreturn({mask=[]}) = 0

[pid 5525] <... read resumed>"", 1024) = 0

[pid 5525] kill(5586, SIGUSR1) = 0

[pid 5586] --- SIGUSR1 {si\_signo=SIGUSR1, si\_code=SI\_USER, si\_pid=5525, si\_uid=1000} ---

[pid 5525] **munmap(0x7f70da823000, 1024) = 0**

[pid 5586] exit\_group(0 <unfinished ...>

[pid 5525**] munmap(0x7f70da7e9000, 128 <unfinished ...>**

[pid 5586] <... exit\_group resumed>) = ?

[pid 5525] <... munmap resumed>) = 0

[pid 5586] +++ exited with 0 +++

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=5586, si\_uid=1000, si\_status=0, si\_utime=8122, si\_stime=4} ---

wait4(5586, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 5586

exit\_group(0) = ?

+++ exited with 0 +++

**Вывод**

В результате проделанной работы я научился реализовывать еще один способ взаимодействия между несколькими процессами. В данном случае нет нужды искусственно связывать между собой child1 и child2, ведь они оба имеют доступ к введенной строке. По моему мнению данный метод является лишь еще одной реализацией связи между процессами, эквивалентный методу, который подразумевал использование каналов.