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: М80-206Б-22

Студент: Ларин И.А.

Преподаватель: Миронов Е.С.

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

Дата: 31.10.23

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.
* int pipe(int \*fd); – создает неименованный канал, у которого первое поле отвечает за чтение, а второе - за запись.
* int execv(const char \*\_\_path, char \*const \*\_\_argv); - предоставляет новой программе список аргументов в виде массива указателей на строки, заканчивающиеся (char \*)0.
* int dup2(int, int); - создает копию файлового дескриптора oldfd *(1 поле)*, используя для нового дескриптора newfd *(2 поле)* файловый дескриптор (они становятся взаимозаменяемыми).
* \_exit(int status); – выходит из процесса с заданным статусом.
* pid\_t wait(int \*status); – приостаналивает выполнение текущего процесса до тех пор, пока дочерний процесс не завершится.
* int read(int fd, void \*buffer, int nbyte); – читает nbyte байтов из файлового дескриптора fd в буффер buffer.
* ssize\_t write(int fd, const void \*buf, size\_t count); **-** записывает количества байт из буфера, указанного buf в файл, на который ссылается файловый дескриптор fd.

Первой строчкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия файла с таким именем на чтение Далее происходит проверка поданного файла на чтение, и если прочитался успешно, создаётся pipe и дочерний процесс (с дальнейшими проверками их создания). Потом происходит перераспределение файловых дескрипторов стандартного ввода будет с pipe1, вывода на файл, ошибок в pipe2 в дочернем процессе. Следующим шагом дочерний процесс запускает программу child.c и обрабатывает свой стандартный ввод (проверяет строку на заглавную букву). Если ошибка, то выводит “invalid string”. В то же время, родительский процесс читает pipe2 и выводит полученные результаты в стандартный поток вывода, кроме того, записывает в pipe1 введённую строку пользователем для проверки. Вся программа сопровождается многочисленными проверками.

**Код программы**

**Main.c**

#include <stdio.h>

#include <unistd.h>

#include <sys/wait.h>

#include <string.h>

#include <fcntl.h>

#include <errno.h>

#include <poll.h>

#include "d\_string.h"

int loop(int pipe\_out, int pipe\_in) {

    errno = 0;

    char\* st = NULL;

    int len = 0;

    int scan\_res;

    do {

       scan\_res = scan\_string(&st, &len, STDIN\_FILENO);

       if (scan\_res && scan\_res != EOF)

           return errno;

       if (write(pipe\_out, st, len\*sizeof(char)) == -1 ||

        write(pipe\_out, "\n", sizeof(char)) == -1)

           return errno;

       if (scan\_res == EOF && len == 0)

           break;

       int poll\_res = poll(&(struct pollfd){.fd=pipe\_in, .events=POLLIN}, 1, 1);

       if (poll\_res == -1)

           return errno;

       if (poll\_res == 1) {

           char\* err\_st = NULL;

           int len\_st = 0;

           if (scan\_string(&err\_st, &len\_st, pipe\_in))

               return errno;

           printf("%s\n", err\_st);

       }

    } while (!scan\_res);

    return errno;

}

int main() {

    errno = 0;

    int fd1[2];

    int fd2[2];

    char\* filename = NULL;

    int filename\_len = 0;

    if (scan\_string(&filename, &filename\_len, STDIN\_FILENO)) {

        if (errno) {

            perror("scan string error");

            return errno;

        }

        return -1;

    }

    int file\_d = creat(filename, S\_IRWXU);

    if (file\_d < 0)

        perror("open error");

    else if (pipe(fd1))

        perror("pipe1 error");

    else if (pipe(fd2))

        perror("pipe2 error");

    if (errno)

        return errno;

    pid\_t id = fork();

    if (id < 0) {

        perror("fork error");

        return errno;

    } else if (id == 0) {// 0 для чтения 1 для записи

        if (dup2(fd1[0], STDIN\_FILENO) == -1)

            perror("dup2 stdin error");

        else if (dup2(file\_d, STDOUT\_FILENO) == -1)

            perror("dup2 stdout error");

        else if (dup2(fd2[1], STDERR\_FILENO) == -1)

            perror("dup2 stderr error");

        else if (close(fd2[0]))

            perror("close pipe2 read error");

        else if (close(fd1[1]))

            perror("close pipe1 write error");

        else {

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

            perror("execl error");

        }

        return errno;

    }

    if (close(fd2[1]))

        perror("close pipe2 write error");

    else if (close(fd1[0]))

        perror("close pipe1 read error");

    else if (close(file\_d))

        perror("close file write error");

    else if (loop(fd1[1], fd2[0]))

        perror("loop error");

    else if (close(fd2[0]))

        perror("close pipe2 read error");

    else if (close(fd1[1]))

        perror("close pipe1 write error");

    else {

        int status;

        waitpid(id, &status, 0);

        return status;

    }

    return errno;

}

Child.c

#include <stdio.h>

#include <unistd.h>

#include <string.h>

#include <errno.h>

#include "d\_string.h"

int is\_upper(char ch) {

    return 'A' <= ch && ch <= 'Z';

}

int main() {

    errno = 0;

    char\* invalid\_string = "invalid string\n";

    char\* st = NULL;

    int len = 0;

    while (!scan\_string(&st, &len, STDIN\_FILENO)) {

        if (is\_upper(st[0])) {

            write(STDOUT\_FILENO, st, len \* sizeof(char));

        }

        else if (write(STDERR\_FILENO, invalid\_string,

                         strlen(invalid\_string)\*sizeof(char)) == -1) {

            perror("write error");

            return errno;

        }

    }

    return errno;

}

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 fd) {

    errno = 0;

    int buf\_size = 1;

    int new\_len = 0;

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

    if (new\_str == NULL)

        return -1;

    char c;

    int res\_read;

    while ((res\_read = read(fd, &c, sizeof(char))) > 0 && c != '\n') {

        new\_str[new\_len++] = 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';

    if (res\_read == -1)

        return errno;

    free(\*str);

    \*str = new\_str;

    \*len = new\_len;

    if (res\_read == 0)

        return EOF;

    return 0;

}

d\_string.h

#ifndef LAB1\_D\_STRING\_H

#define LAB1\_D\_STRING\_H

int scan\_string(char\*\* str, int\* len, int fd);

#endif //LAB1\_D\_STRING\_H

**Протокол работы программы**

**Тестирование:**

ilya@ilya-ilyal:~/osi\_1$ gcc -Wall main.c d\_string.c -o main

ilya@ilya-ilyal:~/osi\_1$ gcc -Wall child.c d\_string.c -o child

ilya@ilya-ilyal:~/osi\_1$ ./main

file.txt

sffsfrsr

invalid string

frcrf

invalid string

frrfcfrc

invalid string

Sfdfvcd

SFFC

CDDCFBJD

fcr

invalid string

CCD

VGTGTRF

ww

invalid string

FRNFRHR

denj

invalid string

**Strace**

ilya@ilya-ilyal:~/osi\_1$ strace -f ./main

execve("./main", ["./main"], 0x7ffefb121d68 /\* 59 vars \*/) = 0

brk(NULL) = 0x55efbe951000

arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7ffc4c3f54d0) = -1 EINVAL (Недопустимый аргумент)

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7fd4b2475000

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=53143, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 53143, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7fd4b2468000

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\"\233}\305\t\5?\344\337^)\350b\231\21\360"..., 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) = 0x7fd4b2200000

mmap(0x7fd4b2228000, 1658880, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x7fd4b2228000

mmap(0x7fd4b23bd000, 360448, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1bd000) = 0x7fd4b23bd000

mmap(0x7fd4b2415000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x214000) = 0x7fd4b2415000

mmap(0x7fd4b241b000, 52816, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7fd4b241b000

close(3) = 0

mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7fd4b2465000

arch\_prctl(ARCH\_SET\_FS, 0x7fd4b2465740) = 0

set\_tid\_address(0x7fd4b2465a10) = 4844

set\_robust\_list(0x7fd4b2465a20, 24) = 0

rseq(0x7fd4b24660e0, 0x20, 0, 0x53053053) = 0

mprotect(0x7fd4b2415000, 16384, PROT\_READ) = 0

mprotect(0x55efbe35a000, 4096, PROT\_READ) = 0

mprotect(0x7fd4b24af000, 8192, PROT\_READ) = 0

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

munmap(0x7fd4b2468000, 53143) = 0

getrandom("\x13\xe4\xdb\x2d\x55\x3d\x16\x28", 8, GRND\_NONBLOCK) = 8

brk(NULL) = 0x55efbe951000

brk(0x55efbe972000) = 0x55efbe972000

**read(0, sdsdsfdsaf**

"s", 1) = 1

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

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

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

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

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

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

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

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

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

**read(0, "\n", 1) = 1**

creat("sdsdsfdsaf", 0700) = 3

**pipe2([4, 5], 0) = 0**

**pipe2([6, 7], 0) = 0**

**clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLDstrace: Process 4897 attached**

, child\_tidptr=0x7fd4b2465a10) = 4897

[pid 4897] set\_robust\_list(0x7fd4b2465a20, 24 <unfinished ...>

[pid 4844] close(7 <unfinished ...>

[pid 4897] <... set\_robust\_list resumed>) = 0

[pid 4844] <... close resumed>) = 0

[pid 4897**] dup2(4, 0 <unfinished ...>**

[pid 4844] close(4 <unfinished ...>

[pid 4897] <... dup2 resumed>) = 0

[pid 4844] <... close resumed>) = 0

[pid 4897**] dup2(3, 1 <unfinished ...>**

[pid 4844] close(3 <unfinished ...>

[pid 4897] <... dup2 resumed>) = 1

[pid 4844] <... close resumed>) = 0

[pid 4897] **dup2(7, 2)** = 2

[pid 4897] close(6) = 0

[pid 4897] close(5 <unfinished ...>

[pid 4844] read(0, <unfinished ...>

[pid 4897] <... close resumed>) = 0

[pid 4897] **execve("./child", ["./child"], 0x7ffc4c3f56a8 /\* 59 vars \*/) = 0**

[pid 4897] brk(NULL) = 0x561cb1034000

[pid 4897] arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7ffe7c6de5d0) = -1 EINVAL (Недопустимый аргумент)

[pid 4897] mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f1f6fcc6000

[pid 4897] access("/etc/ld.so.preload", R\_OK) = -1 ENOENT (Нет такого файла или каталога)

[pid 4897] openat(AT\_FDCWD, "/etc/ld.so.cache", O\_RDONLY|O\_CLOEXEC) = 5

[pid 4897] newfstatat(5, "", {st\_mode=S\_IFREG|0644, st\_size=53143, ...}, AT\_EMPTY\_PATH) = 0

[pid 4897] mmap(NULL, 53143, PROT\_READ, MAP\_PRIVATE, 5, 0) = 0x7f1f6fcb9000

[pid 4897] close(5) = 0

[pid 4897] openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libc.so.6", O\_RDONLY|O\_CLOEXEC) = 5

[pid 4897] **read(5, "\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 4897] pread64(5, "\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 4897] pread64(5, "\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 4897] pread64(5, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\"\233}\305\t\5?\344\337^)\350b\231\21\360"..., 68, 896) = 68

[pid 4897] newfstatat(5, "", {st\_mode=S\_IFREG|0755, st\_size=2216304, ...}, AT\_EMPTY\_PATH) = 0

[pid 4897] pread64(5, "\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 4897] mmap(NULL, 2260560, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 5, 0) = 0x7f1f6fa00000

[pid 4897] mmap(0x7f1f6fa28000, 1658880, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 5, 0x28000) = 0x7f1f6fa28000

[pid 4897] mmap(0x7f1f6fbbd000, 360448, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 5, 0x1bd000) = 0x7f1f6fbbd000

[pid 4897] mmap(0x7f1f6fc15000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 5, 0x214000) = 0x7f1f6fc15000

[pid 4897] mmap(0x7f1f6fc1b000, 52816, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f1f6fc1b000

[pid 4897] close(5) = 0

[pid 4897] mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f1f6fcb6000

[pid 4897] arch\_prctl(ARCH\_SET\_FS, 0x7f1f6fcb6740) = 0

[pid 4897] set\_tid\_address(0x7f1f6fcb6a10) = 4897

[pid 4897] set\_robust\_list(0x7f1f6fcb6a20, 24) = 0

[pid 4897] rseq(0x7f1f6fcb70e0, 0x20, 0, 0x53053053) = 0

[pid 4897] mprotect(0x7f1f6fc15000, 16384, PROT\_READ) = 0

[pid 4897] mprotect(0x561caf7a7000, 4096, PROT\_READ) = 0

[pid 4897] mprotect(0x7f1f6fd00000, 8192, PROT\_READ) = 0

[pid 4897] prlimit64(0, RLIMIT\_STACK, NULL, {rlim\_cur=8192\*1024, rlim\_max=RLIM64\_INFINITY}) = 0

[pid 4897] munmap(0x7f1f6fcb9000, 53143) = 0

[pid 4897] getrandom("\xb2\xd9\xb6\xd8\x13\xd7\x66\x2f", 8, GRND\_NONBLOCK) = 8

[pid 4897] brk(NULL) = 0x561cb1034000

[pid 4897] brk(0x561cb1055000) = 0x561cb1055000

[pid 4897] read(0, sfsd

<unfinished ...>

[pid 4844] <... read resumed>"s", 1) = 1

[pid 4844**] read(0, "f", 1) = 1**

**[pid 4844] read(0, "s", 1) = 1**

**[pid 4844] read(0, "d", 1) = 1**

**[pid 4844] read(0, "\n", 1) = 1**

**[pid 4844] write(5, "sfsd", 4) = 4**

**[pid 4897] <... read resumed>"s", 1) = 1**

**[pid 4844] write(5, "\n", 1) = 1**

**[pid 4897] read(0, <unfinished ...>**

[pid 4844] poll([{fd=6, events=POLLIN}], 1, 1 <unfinished ...>

[pid 4897] <... read resumed>"f", 1) = 1

[pid 4897] **read(0, "s", 1) = 1**

[pid 4897] **read(0, "d", 1) = 1**

**[pid 4897] read(0, "\n", 1) = 1**

**[pid 4897] write(2, "invalid string\n", 15 <unfinished ...>**

[pid 4844] <... poll resumed>) = 1 ([{fd=6, revents=POLLIN}])

[pid 4897] <... write resumed>) = 15

[pid 4897] read(0, <unfinished ...>

[pid 4844] **read(6, "i", 1) = 1**

**[pid 4844] read(6, "n", 1) = 1**

**[pid 4844] read(6, "v", 1) = 1**

**[pid 4844] read(6, "a", 1) = 1**

**[pid 4844] read(6, "l", 1) = 1**

**[pid 4844] read(6, "i", 1) = 1**

**[pid 4844] read(6, "d", 1) = 1**

**[pid 4844] read(6, " ", 1) = 1**

**[pid 4844] read(6, "s", 1) = 1**

**[pid 4844] read(6, "t", 1) = 1**

**[pid 4844] read(6, "r", 1) = 1**

**[pid 4844] read(6, "i", 1) = 1**

**[pid 4844] read(6, "n", 1) = 1**

**[pid 4844] read(6, "g", 1) = 1**

**[pid 4844] read(6, "\n", 1) = 1**

[pid 4844] newfstatat(1, "", {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0), ...}, AT\_EMPTY\_PATH) = 0

**[pid 4844] write(1, "invalid string\n", 15invalid string**

**) = 15**

**[pid 4844] read(0, CVH**

**"C", 1) = 1**

**[pid 4844] read(0, "V", 1) = 1**

**[pid 4844] read(0, "H", 1) = 1**

**[pid 4844] read(0, "\n", 1) = 1**

**[pid 4844] write(5, "CVH", 3) = 3**

**[pid 4897] <... read resumed>"C", 1) = 1**

**[pid 4844] write(5, "\n", 1 <unfinished ...>**

**[pid 4897] read(0, <unfinished ...>**

[pid 4844] <... write resumed>) = 1

[pid 4897] <... read resumed>"V", 1) = 1

[pid 4844] poll([{fd=6, events=POLLIN}], 1, 1 <unfinished ...>

[pid 4897] **read(0, "H", 1) = 1**

**[pid 4897] read(0, "\n", 1) = 1**

[pid 4844] <... poll resumed>) = 0 (Timeout)

**[pid 4897] write(1, "CVH", 3 <unfinished ...>**

**[pid 4844] read(0, <unfinished ...>**

[pid 4897] <... write resumed>) = 3

**Вывод**

Я на практике понял принципы работы с неименованными каналами для межпроцессного взаимодействия, разобрался, как перенаправлять потоки ввода/вывода, а также научился использовать системные вызовы и обращаться с файловыми дескрипторами. В этой работе удалось лучше узнать язык программирования C/C++. Я уверен, что эти знания мне помогут в дальнейшем.