**Московский авиационный институт (Национальный исследовательский университет)**

**Факультет информационных технологий и прикладной математики Кафедра вычислительной математики и программирования**

# Лабораторная работа № 4 по курсу

**«Операционные системы»**

Студент: Соколов Даниил Витальевич

Группа: 8О-207Б Преподаватель: Е. С. Миронов Вариант: 19

Дата:

Оценкa:

Москва, 2022

# Лабораторная работа №4

## Описание

Данная лабораторная работа будет выполняться в ОС Unix.

Процесс – абстракция, описывающая выполняющуюся программу. Программа считывает два файла на открытие для записи в них пользовательских строк дочерними процессами 1 и 2. Далее программа создаёт дочерние процессы 1 и 2 системным вызовом fork(). Взаимодействие между процессами осуществляется с помощью отображения файлов в память. В родительском процессе мы принимаем строки и фильтруем их: с шансом в 80% перекинуть файлы во второй файл, выкидывая все гласные. Отображение файлов в память осуществляется при помощи системного вызова mmap. Возможные системные ошибки обрабатываются с помощью системного вызова exit().

## Исходный код

main.c

#include <stdlib.h>

#include <stdio.h>

#include <unistd.h>

#include <fcntl.h>

#include <string.h>

#include <sys/wait.h>

#include <sys/mman.h>

#include <stdint.h>

pid\_t create\_child(char \*filename, int type)

{

pid\_t pid = fork();

if (pid < 0)

{

perror("Fork err");

exit(1);

}

if (pid == 0)

{

int file\_out = open(filename, O\_WRONLY | O\_CREAT | O\_TRUNC, 0777);

if (file\_out < 0)

{

perror("File err");

exit(1);

}

int new\_out = dup2(file\_out, STDOUT\_FILENO);

close(file\_out);

if (new\_out < 0)

{

perror("Duping child stdout err");

exit(1);

}

if (execlp("./child", "child", type ? "1" : "0", NULL) < 0)

{

perror("Child execl err");

exit(1);

}

}

return pid;

}

char \*ufgets(FILE \*stream)

{

unsigned int maxlen = 128, size = 128;

char \*buffer = (char \*)malloc(maxlen);

if (buffer != NULL) /\* NULL if malloc() fails \*/

{

int ch = EOF;

int pos = 0;

/\* Read input one character at a time, resizing the buffer as necessary \*/

while ((ch = fgetc(stream)) != '\n' && ch != EOF && !feof(stream))

{

buffer[pos++] = ch;

if (pos == size) /\* Next character to be inserted needs more memory \*/

{

size = pos + maxlen;

buffer = (char \*)realloc(buffer, size);

}

}

buffer[pos] = '\0'; /\* Null-terminate the completed string \*/

}

return buffer;

}

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

{

printf("Enter filename for first process\n");

char \*filename1 = ufgets(stdin);

printf("Enter filename for second process\n");

char \*filename2 = ufgets(stdin);

int input = open("./input.txt", O\_RDWR | O\_CREAT | O\_TRUNC, 0777);

if (input < 0) {

perror("Couldn't open input file\n");

exit(1);

}

char \*msg;

printf("Enter strings to process: \n");

while ((msg = ufgets(stdin)) && msg[0] != '\0')

{

msg[strlen(msg)] = '\n';

write(input, msg, sizeof(char) \* strlen(msg));

}

pid\_t pid1, pid2;

pid1 = create\_child(filename1, 0);

pid2 = create\_child(filename2, 1);

close(input);

int statusChild1, statusChild2;

waitpid(pid1, &statusChild1, 0);

if (WIFEXITED(statusChild1))

{

printf("Child 1 exited, returned %d\n", WEXITSTATUS(statusChild1));

}

else

{

fprintf(stderr, "Something is wrong with 1st child process\n");

}

waitpid(pid2, &statusChild2, 0);

if (WIFEXITED(statusChild1))

{

printf("Child 2 exited, returned %d\n", WEXITSTATUS(statusChild2));

}

else

{

fprintf(stderr, "Something is wrong with 2nd child process\n");

}

free(msg);

return 0;

}

..child.c

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <ctype.h>

#include <stdint.h>

#include <fcntl.h>

#include <sys/wait.h>

#include <sys/mman.h>

#include "sys/stat.h"

#include <time.h>

#include <ctype.h>

int checker(char x) {

x = tolower(x);

if (x == 'a' || x == 'e' || x == 'y' || x == 'u' || x == 'o') {

return 1;

} else

return 0;

}

void Reverse(char \*newstring)

{

char \* start = newstring;

char sym;

int len = 0;

char tmp;

while ((sym = \*newstring) != '\n' && (sym = \*newstring) != '\0') {

if (!checker(sym)) {

printf("%c", sym);

}

newstring++;

}

printf("\n");

}

char \*ufgetl(char \*stream)

{

unsigned int maxlen = 128, size = 128;

char \*buffer = (char \*)malloc(maxlen);

if (buffer != NULL) /\* NULL if malloc() fails \*/

{

int ch = EOF;

int pos = 0;

/\* Read input one character at a time, resizing the buffer as necessary \*/

while ((ch = \*(stream + pos)) != '\n' && ch != EOF)

{

buffer[pos++] = ch;

if (pos == size) /\* Next character to be inserted needs more memory \*/

{

size = pos + maxlen;

buffer = (char \*)realloc(buffer, size);

}

}

buffer[pos] = '\0'; /\* Null-terminate the completed string \*/

}

return buffer;

}

void PrintReverse(char \*msg, int type, int count) {

if ((count >= 80 && type) || (count < 80 && !type)) {

Reverse(msg);

}

return;

}

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

{

// type = 0 -> Д.процесс 1, type = 1 -> Д.процесс 2

int type = atoi(argv[1]);

int fd = open("./input.txt", O\_RDWR, 0777);

if (fd < 0) {

perror("Couldn't open input file");

exit(1);

}

struct stat st;

if (fstat(fd, &st) < 0)

{

perror("Couldn't get the input file size");

exit(1);

}

uint8\_t \*byte\_ptr = mmap(NULL,

st.st\_size,

PROT\_READ | PROT\_WRITE, MAP\_SHARED,

fd, 0);

if (byte\_ptr < 0) {

perror("Couldn't mmap file");

exit(1);

}

char \*msg;

int pos = 0;

int count = 0;

srand(time(NULL));

while ( (msg = ufgetl(byte\_ptr + pos)) && msg[0] != '\0' )

{

count = rand() % 100 + 1;

printf("%d\n", count);

PrintReverse(msg, type, count);

pos += strlen(msg);

if (byte\_ptr[pos] == '\n') pos++;

}

if(munmap(byte\_ptr, st.st\_size) != 0){

perror("Couldn't munmap ptr");

exit(1);

}

return 0;

}

## Тестирование программы и вывод команды strace

danchik@danchik-MS-7A20:~/Рабочий стол/repositories/lab04$ strace ./code

execve("./code", ["./code"], 0x7ffcdb80bcb0 /\* 48 vars \*/) = 0

brk(NULL) = 0x558682c48000

arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7ffde74c2f30) = -1 EINVAL (Недопустимый аргумент)

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=78353, ...}) = 0

mmap(NULL, 78353, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7fcb5f14a000

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\t\233\222%\274\260\320\31\331\326\10\204\276X>\263"..., 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) = 0x7fcb5f148000

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\t\233\222%\274\260\320\31\331\326\10\204\276X>\263"..., 68, 880) = 68

mmap(NULL, 2036952, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7fcb5ef56000

mprotect(0x7fcb5ef7b000, 1847296, PROT\_NONE) = 0

mmap(0x7fcb5ef7b000, 1540096, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x25000) = 0x7fcb5ef7b000

mmap(0x7fcb5f0f3000, 303104, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x19d000) = 0x7fcb5f0f3000

mmap(0x7fcb5f13e000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1e7000) = 0x7fcb5f13e000

mmap(0x7fcb5f144000, 13528, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7fcb5f144000

close(3) = 0

arch\_prctl(ARCH\_SET\_FS, 0x7fcb5f149540) = 0

mprotect(0x7fcb5f13e000, 12288, PROT\_READ) = 0

mprotect(0x558681a98000, 4096, PROT\_READ) = 0

mprotect(0x7fcb5f18b000, 4096, PROT\_READ) = 0

munmap(0x7fcb5f14a000, 78353) = 0

fstat(1, {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0), ...}) = 0

brk(NULL) = 0x558682c48000

brk(0x558682c69000) = 0x558682c69000

write(1, "Enter filename for first process"..., 33Enter filename for first process

) = 33

fstat(0, {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0), ...}) = 0

read(0, out1

"out1\n", 1024) = 5

write(1, "Enter filename for second proces"..., 34Enter filename for second process

) = 34

read(0, out2

"out2\n", 1024) = 5

openat(AT\_FDCWD, "./input.txt", O\_RDWR|O\_CREAT|O\_TRUNC, 0777) = 3

write(1, "Enter strings to process: \n", 27Enter strings to process:

) = 27

read(0, rtuh;rtjhgdjh;sdf

"rtuh;rtjhgdjh;sdf\n", 1024) = 18

write(3, "rtuh;rtjhgdjh;sdf\n", 18) = 18

read(0, sfjhg;lkdhfjslkhglkdfkhkdjhkfdklh

"sfjhg;lkdhfjslkhglkdfkhkdjhkfdkl"..., 1024) = 34

write(3, "sfjhg;lkdhfjslkhglkdfkhkdjhkfdkl"..., 34) = 34

read(0, akdfjkdjsklfjklsjlkgksd

"akdfjkdjsklfjklsjlkgksd\n", 1024) = 24

write(3, "akdfjkdjsklfjklsjlkgksd\n", 24) = 24

read(0, ashdjhg;lfaglkhlfdh

"ashdjhg;lfaglkhlfdh\n", 1024) = 20

write(3, "ashdjhg;lfaglkhlfdh\n", 20) = 20

read(0, asghjl;kfjahklfjalhjklte

"asghjl;kfjahklfjalhjklte\n", 1024) = 25

write(3, "asghjl;kfjahklfjalhjklte\n", 25) = 25

read(0, ^C0x558682c48740, 1024) = ? ERESTARTSYS (To be restarted if SA\_RESTART is set)

strace: Process 8978 detached

danchik@danchik-MS-7A20:~/Рабочий стол/repositories/lab04$ cat out1 out2

55

qwrt

35

kdfh

95

64

rphkprjhrpj

41

dhpkdh

58

sgrgr[pg

65

fdpgkpdkb

96

85

55

35

95

phkh

64

41

58

65

96

s[gs[gf

85

gg

danchik@danchik-MS-7A20:~/Рабочий стол/repositories/lab04$

## Вывод

В лабораторной работе №4 я познакомился с таким способом взаимодействия процессов как отображение файлов в память. Основным преимуществом такого метода является скорость.