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: М80-206Б-22

Студент: Бурунов М.А.

Преподаватель: Миронов Е.С.

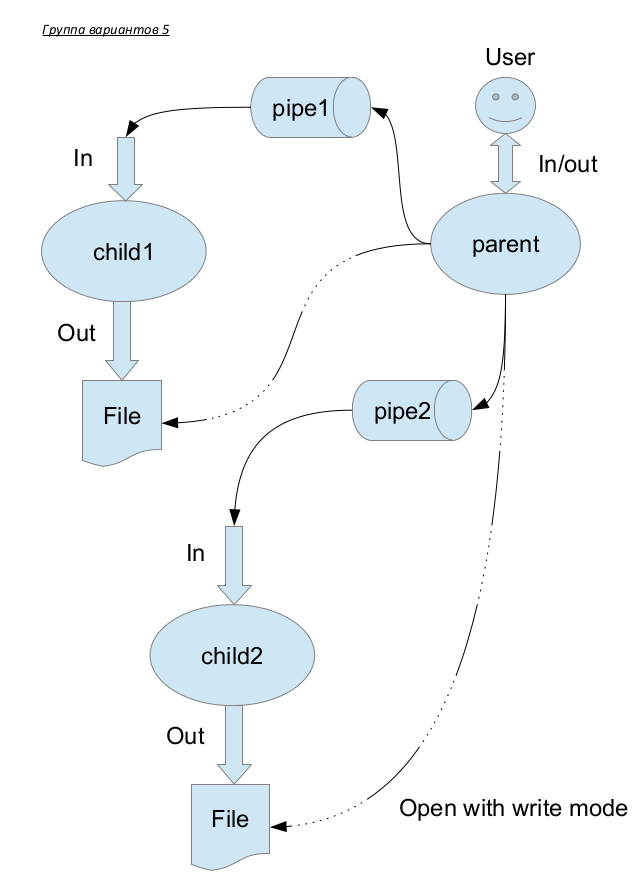
Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

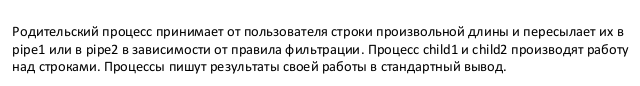
Дата: 01.03.24

Москва, 2024

**Постановка задачи**

Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия файла с таким именем на чтение. Стандартный поток ввода дочернего процесса переопределяется открытым файлом. Дочерний процесс читает команды из стандартного потока ввода. Родительский процесс выводит результат в свой стандартный поток вывода. Родительский и дочерний процесс должны быть представлены разными программами. Взаимодействие между процессами осуществляется через системные сигналы/события и/или через отображаемые файлы (memory-mapped files).

****



**Вариант 19.**  
 Правило фильтрации: с вероятностью 80% строки отправляются в pipe1, иначе в pipe2. Дочерние процессы удаляют гласные из строк.

**Общий метод и алгоритм решения**

Использованные системные вызовы:

* pid\_t fork(void); – создает дочерний процесс, возвращает PID дочернего процесса, а процессу потомку возвращается 0, а в случае ошибки -1.
* ssize\_t write(int fd, const void buf[count], size\_t count) - записывает size\_t count байт в указанный файловый дескриптор fd, после завершения возвращает количество записанных байтов, а в случае ошибки возвращает -1.
* int open(const char \*pathname, int flags, mode\_t mode) - открывает и создает файл(при указании нужного флага), возвращает файловый дескриптор, а в случае ошибки -1.
* int close(int fd) - закрывает файловый дескриптор fd.
* int dup2(int oldfd, int newfd) - дублирует файловый дескриптор newfd на место дескриптора oldfd, возвращает новый дескриптор, а в случае ошибки -1.
* int execv(const char \*path, char \*const argv[]) - загружает на исполнение указанную программу (необходимо указать полный путь) и аргументы командной строки в виде массива указателей на эти строки.
* int **kill**(pid\_t pid, int sig) - посылает сигнал процессу или выводит список допустимых сигналов.
* void (\*signal (int signal, void (\*sigfunc) (int func)))(int) - дает указание выполнить функцию, на которую указывает sigfunc, в случае получения сигнала signal.
* void\* **mmap**(void \*addr, size\_t length, int prot, int flags, int fd, off\_t offset) - создает отображение в виртуальном адресном пространстве вызывающего процесса. При удачном отображении возвращает указатель на область памяти с отображенными данными.
* int **munmap**(void \*addr, size\_t length) - используется для удаления ранее сопоставленной области памяти из виртуального адресного пространства вызывающего процесса.
* int **ftruncate**(int, off\_t) - приводит файл к заданному размеру.
* int **shm\_open**(const char \*, int, ...) - используется для создания или открытия разделяемого сегмента памяти (область памяти, используемой несколькими процессами для обмена данными). Возвращает файловый дескриптор.
* int **shm\_unlink**(MEMORY\_NAME) - используется для удаления именованного сегмента разделяемой памяти.
* void exit(int status) - приводит к удачному/неудачному завершению программы.

После запуска программа потребует ввести имена файлов (с расширением) c помощью функции char\* inputing(). Данная функция позволяет ввести строчку любой длины, ввод будет осуществляться, пока не будет введен символ переноса строки (‘\n’).

После этого вызов open получает в аргументы уже введенную нами и обработанную строчку и открывает файл с этим именем, а если такого файла нет - он будет создан.

Далее с помощью shm\_open создаются два именованных сегмента разделяемой памяти, размер которых указывается отдельно с помощью ftruncate. Полученный дескриптор впоследствии используется при создании отображения файла в адресное пространство процесса - с помощью mmap, возвращающей указатель на область памяти для работы с отображенными данными.

Функция “process\_creation” является оболочкой fork(), в которой одновременно с вызовом происходит проверка на ошибки, а в случае ошибки программа аварийно завершается. С помощью данной функции происходит порождение дочерних процессов.

Если созданный процесс - ребенок(“fork()” вернул 0), то программа с помощью вызова dup2 подменяет для дочернего процесса стандартный поток вывода на “f1\_output” (открытый в начале программы файл). После этого 1-ый дочерний процесс с помощью системного вызова execv загружает на исполнение указанную программу child.c и передает к ней аргументы командной строки в виде массива указателей на эти строки, инициализрованные в самом начале программы.

Для 2-го дочернего процесса все происходит аналогично.

Родитель, в свою очередь, запускает бесконечный цикл для ввода строк, количество которых ограничено количеством выделенного места под именнованные сегменты разделяемой памяти т.к. именно туда посимвольно записываются вводимые строки с помощью mmap (конкретнее - с помощью указателя на отображенную в адресное пространство процесса память). В случае переполнения возникает соответствующая ошибка. Как только итератор доходит до “\n” - родитель отправляет пользовательский системный сигнал SIGUSR1 одному из дочерних процессов, что позволяет дочернему процессу взять и обработать только что введенную строку. Поэтому с точки зрения межпроцессорного взаимодействия - ввод построчный. Вероятность отправки разным дочерним процессам соблюдается с помощью функции int probability().

Обработка строк дочерним процессом происходит с помощью функции void writer(). Эта функция вызывается при получении сигнала SIGUSR1 от родителя. Внутри этой функции происходит посимвольное считываение строки c именнованого разделяемоего сегмента памяти, отраженного в адресное пространство дочернего процесса т.е. считывание строки происходит при помощи указателя от mmap. Опять же, считывание происходит до “\n”, далее строки изменяются согласно заданию с помощью функции bool vowel\_del(char \*\*output\_string, char\* input\_string, int len).

Для завершения работы с программой, нужно ввести EOF(Ctrl+D).

В конце программы происходит отправка пользовательских системных сигналов SIGUSR2 дочерним процессам для завершения их работы. Происходит закрытие файловых дескрипторов, удаление именованных сегментов разделяемой памяти (с помощью shm\_unlink), а также (с помощью munmap) происходит удаление существующих отображений (удаление сопоставленной области памяти из виртуального адресного пространства вызывающего процесса).

**Код программы**

**main.c**

#include <stdio.h>

#include <fcntl.h> //files

#include <stdlib.h> //malloc, srand, rand

#include <stdbool.h>

#include <unistd.h>

#include <sys/types.h> //pid\_t, ftruncate

#include <signal.h> // kill

#include <time.h> //time(NULL)

#include <sys/mman.h>

#include "stddef.h"

#include <string.h>

#include <sys/stat.h>

#define MEMORY\_NAME1 "fd\_virt1"

#define MEMORY\_NAME2 "fd\_virt2"

char\* inputing() { // fixed inputing

int len = 0;

int capacity = 10; // Начальная емкость

char \*s = (char\*) malloc(capacity \* sizeof(char)); // Выделяем начальную память

if (s == NULL) {

return NULL; // Проверка выделения памяти

}

char c = getchar();

while (c != '\n') {

if (len >= capacity - 1) { // Проверка, нужно ли увеличить буфер

capacity \*= 2;

char\* temp = (char\*) realloc(s, capacity \* sizeof(char)); // Пытаемся увеличить буфер

if (temp == NULL) { // Проверка удачного realloc

free(s);

return NULL;

}

s = temp;

}

s[len++] = c;

c = getchar();

}

s[len] = '\0';

return s;

}

int process\_creation(){

pid\_t pid = fork();

if (pid == -1){

perror("Call fork was ended with erorr: ");

exit(EXIT\_FAILURE);

}

return pid;

}

int probability(){ // from lab1

srand(time(NULL)); //инициализация генератора случайных чисел и установка текущего времени в качестве его базы

int a =rand()%10+1; //случайные числа от 1 до 10

if(a<=8){

return 1;

} else{

return 2;

}

}

int main() {

char \*args1[] = {"child", "123", NULL};

char \*args2[] = {"child", "456", NULL};

write(STDOUT\_FILENO, "Enter the first filename with file extension(.txt or .doc or .rtf): ", 68);

char \* Filename\_1 = NULL;

Filename\_1 = inputing();

if (Filename\_1 == NULL) {

perror ("Inputing error:");

exit(EXIT\_FAILURE);

}

int f1\_output = open(Filename\_1, O\_WRONLY| O\_CREAT | O\_TRUNC , S\_IWUSR);

free(Filename\_1);

if (f1\_output == -1) {

perror ("Can't open the file:");

exit(EXIT\_FAILURE);

}

write(STDOUT\_FILENO, "Enter the second filename with file extension(.txt or .doc or .rtf): ", 69);

char \*Filename\_2 = NULL;

Filename\_2 = inputing();

if (Filename\_2 == NULL) {

perror ("Inputing error:");

exit(EXIT\_FAILURE);

}

int f2\_output = open(Filename\_2, O\_WRONLY| O\_CREAT | O\_TRUNC, S\_IWUSR);

free(Filename\_2);

if (f2\_output == -1) {

perror("Can't open the file:");

exit(EXIT\_FAILURE);

}

int fd1 = shm\_open(

MEMORY\_NAME1,

O\_CREAT | O\_RDWR | O\_TRUNC,

0777);

ftruncate(fd1 , 1000\*sizeof(char));

char \*addr1 = mmap(NULL, 1000\*sizeof(char), PROT\_WRITE | PROT\_READ , MAP\_SHARED ,fd1,0);

if (addr1 == (void\*)-1){

perror("mmap1 error: ");

exit(EXIT\_FAILURE);

}

int fd2 = shm\_open(

MEMORY\_NAME2,

O\_CREAT | O\_RDWR | O\_TRUNC,

0777);

ftruncate (fd2 , 1000\*sizeof(char));

char \*addr2 = mmap(NULL, 1000\*sizeof(char), PROT\_WRITE | PROT\_READ , MAP\_SHARED ,fd2,0);

if (addr2 == (void\*)-1){

perror("mmap2 error: ");

exit(EXIT\_FAILURE);

}

pid\_t pid\_1 = process\_creation();

if (pid\_1 == 0)

{ // the 1st child

if (dup2(f1\_output, STDOUT\_FILENO) == -1){

perror ("dup2 erorr: ");

exit(EXIT\_FAILURE);

}

if(execv("./child", args1) == -1){

perror("execv1 erorr: ");

exit(EXIT\_FAILURE);

}

}else

{ //parent

pid\_t pid\_2 = process\_creation();

if (pid\_2 == 0)

{ // the 2st child

if (dup2(f2\_output, STDOUT\_FILENO) == -1){

perror ("dup2 erorr: ");

exit(EXIT\_FAILURE);

}

if(execv("./child", args2) == -1){

perror("execv2 erorr: ");

exit(EXIT\_FAILURE);

}

}else

{ //parent

char symbol;

int iterator\_1 = 0;

int iterator\_2 = 0;

while(true){

int prob\_res=probability();

if (symbol == EOF){

break;

}

if ((iterator\_1 == 1000\*sizeof(char)) || (iterator\_2 == 1000\*sizeof(char))){

perror("increase the size of the memory mapped file (files): ");

exit(EXIT\_FAILURE);

}

while(((symbol = getchar()) != EOF) && (iterator\_1 != 1000\*sizeof(char)) && (iterator\_2 != 1000\*sizeof(char))){

if (prob\_res==1){

addr1[iterator\_1] = (char)symbol;

iterator\_1 ++;

} else {

addr2[iterator\_2] = (char)symbol;

iterator\_2 ++;

}

if (symbol == '\n'){

if (prob\_res==1){

kill(pid\_1, SIGUSR1);

break;

}

else{

kill(pid\_2, SIGUSR1);

break;

}

}

}

}

kill(pid\_1, SIGUSR2);

kill(pid\_2, SIGUSR2);

close(fd1);

close(fd2);

close(f1\_output);

close(f2\_output);

if(shm\_unlink(MEMORY\_NAME1) == -1){

perror("shm\_unlink(MEMORY\_NAME1) error: ");

exit(EXIT\_FAILURE);

}

if(shm\_unlink(MEMORY\_NAME2) == -1){

perror("shm\_unlink(MEMORY\_NAME2) error: ");

exit(EXIT\_FAILURE);

}

if(munmap(addr1, 1000\*sizeof(char)) == -1){

perror("munmap1 error:");

exit(EXIT\_FAILURE);

}

if(munmap(addr2, 1000\*sizeof(char)) == -1){

perror("munmap2 error:");

exit(EXIT\_FAILURE);

}

}

}

}

**child.c**

#include <stdio.h>

#include <fcntl.h> //files

#include <stdlib.h> //malloc, srand, rand

#include <stdbool.h>

#include <unistd.h>

#include <sys/types.h> //pid\_t, ftruncate

#include <signal.h> // kill

#include <time.h> //time(NULL)

#include <sys/mman.h>

#include "stddef.h"

#include <string.h>

#include <sys/stat.h>

#define MEMORY\_NAME1 "fd\_virt1"

#define MEMORY\_NAME2 "fd\_virt2"

char \*addr\_global;

int flag = 0;

int i\_global = 0;

bool vowel\_del(char \*\*output\_string, char\* input\_string, int len){ // from lab1

char tmp[len+1];

for(int i=0; i<len;++i){

if ((input\_string[i] == 'a' || input\_string[i] == 'e' || input\_string[i] == 'i' || input\_string[i] == 'o' || input\_string[i] == 'u'

|| input\_string[i] == 'y' || input\_string[i] == 'A' || input\_string[i] == 'E' || input\_string[i] == 'I'

|| input\_string[i] == 'O' || input\_string[i] == 'U' || input\_string[i] == 'Y')){

tmp[i]=' ';

}else {

tmp[i] = input\_string[i];

}

}

tmp[len]='\0';

free(\*output\_string);

\*output\_string=tmp;

return true;

}

void writer() {

int c;

char\* output\_string = NULL;

char\* input\_string = NULL;

int input\_string\_size = 0;

for (int i = i\_global; i < 1000 \* sizeof(char); i++) {

c = addr\_global[i];

input\_string = realloc(input\_string, (input\_string\_size + 2) \* sizeof(char));

if ((c != '\n') || (flag != 0)) {

input\_string[input\_string\_size] = c;

input\_string[input\_string\_size + 1] = '\0';

input\_string\_size++;

}

if (c == '\n') {

flag++;

i\_global = i + 1;

break;

}

}

if (string\_invert(&output\_string, input\_string, strlen(input\_string)) == 0) {

write(STDOUT\_FILENO, "String\_invert Error! ", 21);

} else {

write(STDOUT\_FILENO, output\_string, input\_string\_size \* sizeof(char));

}

free(input\_string);

}

void finish(){

exit(EXIT\_SUCCESS);

}

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

int arg = atoi(argv[1]);

int shm\_fd = 0;

if (arg == 123) {

shm\_fd = shm\_open(

MEMORY\_NAME1,

O\_RDWR,

S\_IRUSR);

}else{

shm\_fd = shm\_open(

MEMORY\_NAME2,

O\_RDWR,

S\_IRUSR);

}

char \*addr = mmap(NULL, 1000\*sizeof(char), PROT\_WRITE | PROT\_READ , MAP\_SHARED , shm\_fd, 0);

if (addr == (void\*)-1){

perror("mmap (child) error: ");

exit(EXIT\_FAILURE);

}

addr\_global = addr;

while (true)

{

signal (SIGUSR1, writer);

signal (SIGUSR2, finish);

}

}

**Протокол работы программы**

**Тестирование:**

vimi@HOME-PC:/mnt/c/Documents and Settings/vimi/Desktop/labsOS/3/programs$ ./main

Enter the first filename with file extension(.txt or .doc or .rtf): 1

Enter the second filename with file extension(.txt or .doc or .rtf): 2

one

two

three

four

five

vimi@HOME-PC:/mnt/c/Documents and Settings/vimi/Desktop/labsOS/3/programs$ cat 1.txt

cat: 1.txt: No such file or directory

vimi@HOME-PC:/mnt/c/Documents and Settings/vimi/Desktop/labsOS/3/programs$ cat 1

tw thr

f r

f v

vimi@HOME-PC:/mnt/c/Documents and Settings/vimi/Desktop/labsOS/3/programs$ cat 2

n vimi@HOME-PC:/mnt/c/Documents and Settings/vimi/Desktop/labsOS/3/programs$ ===============================================================================

vimi@HOME-PC:/mnt/c/Documents and Settings/vimi/Desktop/labsOS/3/programs$ ./main

Enter the first filename with file extension(.txt or .doc or .rtf): 1

Enter the second filename with file extension(.txt or .doc or .rtf): 2

hey

wait. I've gonna new complaint

Forever in debt to your priceless advice

She eyes me like a pieces when

I am week

vimi@HOME-PC:/mnt/c/Documents and Settings/vimi/Desktop/labsOS/3/programs$ cat 1

h w t. 'v g nn n w c mpl nt

F r v r n d bt t r pr c l ss dv c

Sh s m l k p c s wh n

m w k

vimi@HOME-PC:/mnt/c/Documents and Settings/vimi/Desktop/labsOS/3/programs$ cat 2

vimi@HOME-PC:/mnt/c/Documents and Settings/vimi/Desktop/labsOS/3/programs$

**Strace:**

execve("./main", ["./main"], 0x7ffc618760c0 /\* 35 vars \*/) = 0

brk(NULL) = 0x55975ad52000

arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7ffec4654c10) = -1 EINVAL (Invalid argument)

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1,

0) = 0x7fc0d01ef000

access("/etc/ld.so.preload", R\_OK) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/etc/ld.so.cache", O\_RDONLY|O\_CLOEXEC) = 3

newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=19095, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 19095, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7fc0d01ea000

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 =\340\2563\265?\356\25x\261\27\313A#\350"..., 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) =

0x7fc0cffc2000

mmap(0x7fc0cffea000, 1658880, PROT\_READ|PROT\_EXEC,

MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x7fc0cffea000

mmap(0x7fc0d017f000, 360448, PROT\_READ,

MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1bd000) = 0x7fc0d017f000

mmap(0x7fc0d01d7000, 24576, PROT\_READ|PROT\_WRITE,

MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x214000) = 0x7fc0d01d7000

mmap(0x7fc0d01dd000, 52816, PROT\_READ|PROT\_WRITE,

MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7fc0d01dd000

close(3) = 0

mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1,

0) = 0x7fc0cffbf000

arch\_prctl(ARCH\_SET\_FS, 0x7fc0cffbf740) = 0

set\_tid\_address(0x7fc0cffbfa10) = 23902

set\_robust\_list(0x7fc0cffbfa20, 24) = 0

rseq(0x7fc0cffc00e0, 0x20, 0, 0x53053053) = 0

mprotect(0x7fc0d01d7000, 16384, PROT\_READ) = 0

mprotect(0x55975ab66000, 4096, PROT\_READ) = 0

mprotect(0x7fc0d0229000, 8192, PROT\_READ) = 0

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

0

munmap(0x7fc0d01ea000, 19095) = 0

write(1, "Enter the first filename with fi"..., 68Enter the first filename with file extension(.txt or

.doc or .rtf): ) = 68

getrandom("\x8c\x4f\x49\xdf\x0e\x4f\x65\x98", 8, GRND\_NONBLOCK) = 8

brk(NULL) = 0x55975ad52000

brk(0x55975ad73000) = 0x55975ad73000

newfstatat(0, "", {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0x4), ...},

AT\_EMPTY\_PATH) = 0

read(0, 1.txt

"1.txt\n", 1024) = 6

openat(AT\_FDCWD, "1.txt", O\_WRONLY|O\_CREAT|O\_TRUNC, 0200) = 3

write(1, "Enter the second filename with f"..., 69Enter the second filename with file extension(.txt

or .doc or .rtf): ) = 69

read(0, 2.txt

"2.txt\n", 1024) = 6

openat(AT\_FDCWD, "2.txt", O\_WRONLY|O\_CREAT|O\_TRUNC, 0200) = 4

**openat(AT\_FDCWD, "/dev/shm/fd\_virt1",**

**O\_RDWR|O\_CREAT|O\_TRUNC|O\_NOFOLLOW|O\_CLOEXEC, 0777) = 5**

ftruncate(5, 1000) = 0

**mmap(NULL, 1000, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 5, 0) = 0x7fc0d0228000**

**openat(AT\_FDCWD, "/dev/shm/fd\_virt2",**

**O\_RDWR|O\_CREAT|O\_TRUNC|O\_NOFOLLOW|O\_CLOEXEC, 0777) = 6**

ftruncate(6, 1000) = 0

**mmap(NULL, 1000, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 6, 0) = 0x7fc0d01ee000**

clone(child\_stack=NULL,

flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD,

child\_tidptr=0x7fc0cffbfa10) = 23957

clone(child\_stack=NULL,

flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD,

child\_tidptr=0x7fc0cffbfa10) = 23958

read(0, piu piu piu

"piu piu piu\n", 1024) = 12

kill(23957, SIGUSR1) = 0

read(0, pu pu pu

"pu pu pu\n", 1024) = 9

kill(23957, SIGUSR1) = 0

read(0, "", 1024) = 0

kill(23957, SIGUSR2) = 0

kill(23958, SIGUSR2) = 0

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=23957, si\_uid=1000,

si\_status=0, si\_utime=215, si\_stime=525} ---

close(5) = 0

close(6) = 0

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=23958, si\_uid=1000,

si\_status=0, si\_utime=216, si\_stime=523} ---

close(3) = 0

close(4) = 0

unlink("/dev/shm/fd\_virt1") = 0

unlink("/dev/shm/fd\_virt2") = 0

**munmap(0x7fc0d0228000, 1000) = 0**

**munmap(0x7fc0d01ee000, 1000) = 0**

exit\_group(0) = ?

+++ exited with 0 +++

**Вывод**

В результате проделанной работы я научился реализовывать еще один способ взаимодействия между несколькими процессами. Работа с пользовательскими системными сигналами и memory-mapped файлами действительно является отличной альтернативой pipe, но хочу отметить, что, в сравнении с pipe, реализовать программу было гораздо сложнее и это потребовало много времени. Уверен, что эти знания пригодятся в дальнейшем.