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: Сарайкин Н.С.

Преподаватель: Миронов Е.С.

Оценка: _____

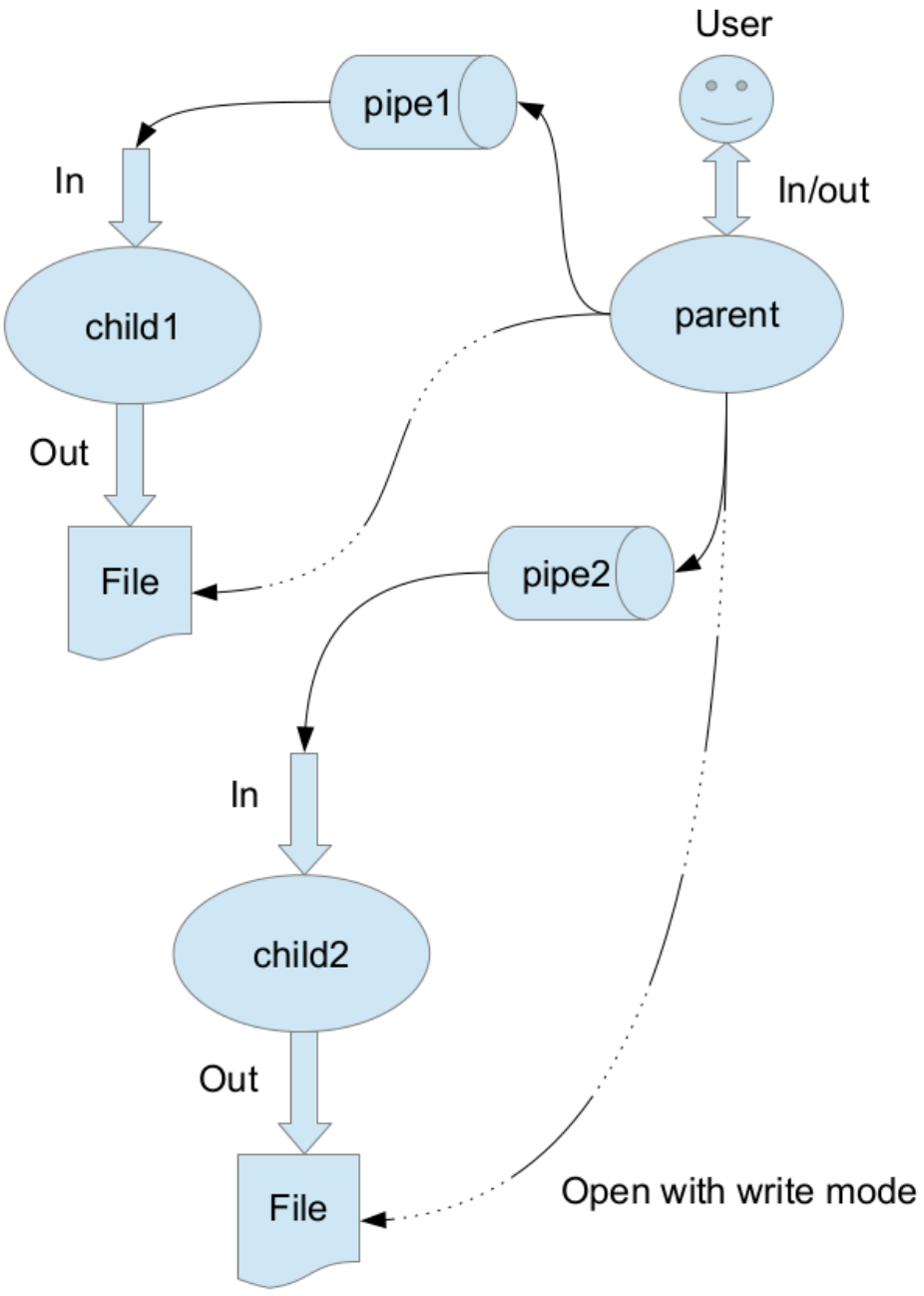
Дата: 16.12.23

Москва, 2023

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).

Группа вариантов 5



Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1 или в pipe2 в зависимости от правила фильтрации. Процесс child1 и child2 производят работу над строками. Процессы пишут результаты своей работы в стандартный вывод.

Вариант 22.

Правило фильтрации: с вероятностью 80% строки отправляются в pipe1, иначе в pipe2. Дочерние процессы инвертируют строки.

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) с 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, инициализированные 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 с 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 инвертируются согласно заданию с помощью функции bool string_invert(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 ((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;
                }
            }
        }
        if (symbol == EOF){
            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 string_invert(char **output_string, char* input_string, int len){ // from lab1
    char tmp[len+1];
```

```
    for(int i=0; i<len;++i){
        tmp[len-1-i]=input_string[i];
    }
    tmp[len]='\0';
    free(*output_string);
    *output_string=tmp;
    return true;
}

void writer() {
    char 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(){
    if(munmap(addr_global, 1000*sizeof(char)) == -1){
        perror("munmap(child) error:");
        exit(EXIT_FAILURE);
    }
    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:

```
mattrrixwsl@DESKTOP-HRTTO4C:/mnt/c/users/Никита/Desktop/Projects/labs3sem/lab3/programs$ ls
child.c  main.c
mattrrixwsl@DESKTOP-HRTTO4C:/mnt/c/users/Никита/Desktop/Projects/labs3sem/lab3/programs$ gcc main.c -o main
mattrrixwsl@DESKTOP-HRTTO4C:/mnt/c/users/Никита/Desktop/Projects/labs3sem/lab3/programs$ gcc child.c -o child
mattrrixwsl@DESKTOP-HRTTO4C:/mnt/c/users/Никита/Desktop/Projects/labs3sem/lab3/programs$ ./main
Enter the first filename with file extension(.txt or .doc or .rtf): 1.txt
Enter the second filename with file extension(.txt or .doc or .rtf): 2.txt
one
two
three
four
five
six
```

```
seven
eight
nine
ten
mattrrixwsl@DESKTOP-HRTTO4C:/mnt/c/users/Никита/Desktop/Projects/labs3sem/lab3/progra
ms$ ls
1.txt  2.txt  child  child.c  main  main.c
mattrrixwsl@DESKTOP-HRTTO4C:/mnt/c/users/Никита/Desktop/Projects/labs3sem/lab3/progra
ms$ cat 1.txt
owt
eerht
ruof
evif
neves
enin
netmattrrixwsl@DESKTOP-HRTTO4C:/mnt/c/users/Никита/Desktop/Projects/labs3sem/lab3/prog
rams$ cat 2.txt
eno
xis
thgie
```

==========

```
mattrrixwsl@DESKTOP-HRTTO4C:/mnt/c/users/Никита/Desktop/Projects/labs3sem/lab3/programs
$ rm *.txt
mattrrixwsl@DESKTOP-HRTTO4C:/mnt/c/users/Никита/Desktop/Projects/labs3sem/lab3/programs
$ ./main
Enter the first filename with file extension(.txt or .doc or .rtf): 1.txt
Enter the second filename with file extension(.txt or .doc or .rtf): 2.txt
smth with spacing
and smth else
mattrrixwsl@DESKTOP-HRTTO4C:/mnt/c/users/Никита/Desktop/Projects/labs3sem/lab3/programs
$ cat 1.txt
gnicaps htiw htms
esle htms
dnamattrrixwsl@DESKTOP-HRTTO4C:/mnt/c/users/Никита/Desktop/Projects/labs3sem/lab3/progr
ams$ cat 2.txt
mattrrixwsl@DESKTOP-HRTTO4C:/mnt/c/users/Никита/Desktop/Projects/labs3sem/lab3/programs
$
```

==========

```
mattrrixwsl@DESKTOP-HRTTO4C:/mnt/c/users/Никита/Desktop/Projects/labs3sem/lab3/programs
$ rm *.txt
mattrrixwsl@DESKTOP-HRTTO4C:/mnt/c/users/Никита/Desktop/Projects/labs3sem/lab3/programs
$ ./main
Enter the first filename with file extension(.txt or .doc or .rtf): print1.doc
```

```
Enter the second filename with file extension(.txt or .doc or .rtf): print2.rtf
looooooooong string with spaaaaaaaaaacing
one
two
three
four
mattrrixwsl@DESKTOP-HRTTO4C:/mnt/c/users/Никита/Desktop/Projects/labs3sem/lab3/programs
$ cat print1.doc
eno
owt
eerht
ruofmattrrixwsl@DESKTOP-HRTTO4C:/mnt/c/users/Никита/Desktop/Projects/labs3sem/lab3/progr
ams$ cat print2.rtf
gnicaaaaaaaaaaps htiw gnirts gnoooooooool
```

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.