Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №4 по курсу «Операционные сети»
Тема: Файловые системы. Отображаемые файлы.

Студент: Д. С. Ляшун
Преподаватель: Е. С. Миронов
Группа: М8О-207Б
Дата:
Оценка:
Подпись:

Москва, 2020

1 Постановка задачи

Цель работы: Приобретение практических навыков в принципах работы с файловыми системами, обеспечении обмена данных между процессами посредством технологии «File mapping».

Задание: Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программа (основной процесс) должен создать для решение задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или через отображаемые файлы (memory-mapped files).

Вариант №2-10.

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. Стандартный поток вывода дочернего процесса перенаправляется в pipe1. Родительский процесс читает из pipe1 и прочитанное выводит в свой стандартный поток вывода. Родительский и дочерний процесс должны быть представлены разными программами.

В файле записаны команды вида: «число<endline>». Дочерний процесс производит проверку этого числа на простоту. Если число составное, то дочерний процесс пишет это число в стандартный поток вывода. Если число отрицательное или простое, то тогда дочерний и родительский процессы завершаются.

2 Алгоритм решения

Алгоритм решения соответствует программе из Лабораторной работы №2, при этом обмен данными между процессами будет реализован посредством общих отображаемых файлов, а взаимодействие – посредством обмена сигналами – SIGUSR1 и SIGUSR2. SIGUSR2 будет передавать дочерний процесс родительскому в двух случаях – когда он запустился в первый раз и произвел свою первоочередную настройку (добавил набор сигналов в множество блокирующих, чтобы они не убивали его при получении), а также когда он завершает свою работу в случае обработки простого числа и достижения конца файла с входными данными. SIGUSR1 будет передаваться родительским процессом, когда он произвел чтение и вывод данных из общего отображаемого файла, т.е. тем самым он дает команду загружать еще данные дочернему процессу, который в свою очередь передает SIGUSR1 родительскому процессу при заполнении общего отображаемого файла с данными.

Для осуществления работы с сигналами, а также общими отображаемыми файлами будут использоваться следующие системные вызовы:

1. int shm_open(const char *name, int oflag, mode_t mode) – создаёт и открывает в режиме oflag и правами доступа mode объект общей памяти с именем name. Функция возвращает неотрицательный дескриптор файла, в случае ошибки – -1.

2. int shm_unlink(const char *name) – удаляет объект общей памяти с именем name, созданный ранее с помощью shm_open. Функция при успешном выполнении возвращает 0, иначе – -1.

3. void * mmap(void *start, size_t length, int prot , int flags, int fd, off_t offset) – отражает length байтов, начиная со смещения offset файла (или другого объекта), определенного файловым описателем fd, в память, начиная с адреса start. Аргумент prot описывает желаемый режим защиты памяти, flags задает тип отражаемого объекта. Функция возвращает указатель на область с отраженными данными.

4. int munmap(void *start, size_t length) – удаляет все отражения из заданной области памяти start размера length байт, после чего все ссылки на данную область будут вызывать ошибку "неправильное обращение к памяти". При успешном выполнении функция возвращает 0, иначе – -1.

5. int sigemptyset(sigset_t *set) – инициализирует набор сигналов, указанный в set, и очищает его от всех сигналов. При успешном выполнении функция возвращает 0, иначе – -1.

6. int sigaddset(sigset_t *set, int signum) – добавляет сигнал signum в набор сигналов set. При успешном выполнении функция возвращает 0, иначе – -1.

7. int sigprocmask(int how, const sigset_t *set, sigset_t *oldset) – изменяет список блокированных сигналов потока в зависимости от значения how (обьединение, удаление, присваивание): происходит работа с указанной группой сигналов set, старый набор сигналов потока сохраняется в oldset. При успешном выполнении функция возвращает 0, иначе – -1.

8. int sigwait(const sigset_t *set, int *sig) – приостанавливает выполнения вызвавшего его потока до тех пор, пока не будет получен сигнал из набора сигналов set, который сохранится в sig. При успешном выполнении функция возвращает 0, иначе – -1.

9. int kill(pid_t pid, int sig) – посылает сигнал sig процессу с номером pid. При успешном выполнении функция возвращает 0, иначе – -1.

3 Листинг программы

Исходный код main.c:

```
#include <signal.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#define check_ok(VALUE, OKVAL, MSG) if (VALUE != OKVAL) { printf("%s\n", MSG); return 1; }
#define check_wrong(VALUE, WRONGVAL, MSG) if (VALUE == WRONGVAL) { printf("%s\n", MSG); return 1; }
char* SHARED_FILE_NAME = "pipe";
const int SIZE_SHARED_FILE = 12;
int main()
{
    /* Work with input data file */
    char input_file[260];
    printf("Input file name: ");
    fgets(input_file, 260, stdin);
    if (input_file[strlen(input_file)-1] == '\n') // cause fgets adding in string \n
    {
        input_file[strlen(input_file)-1] = '\0';
    }
    FILE* input = fopen(input_file, "r");
    check_wrong(input, NULL, "Error creating input data file!");

    /* Work with shared file */
    int sf_d = shm_open(SHARED_FILE_NAME, O_RDWR | O_CREAT, S_IRWXU);
    check_wrong(sf_d, -1, "Error creating shared data file!");
    check_ok(ftruncate(sf_d, SIZE_SHARED_FILE), 0, "Error changing size of shared data file!");
    char* data = mmap(NULL, SIZE_SHARED_FILE, PROT_READ | PROT_WRITE, MAP_SHARED, sf_d, 0);
    check_wrong(data, MAP_FAILED, "Error mapping shared data file!");
    for (int i = 0; i < SIZE_SHARED_FILE; ++i)
    {
        data[i] = '\0';
    }

    /* Work with signal variable */
    sigset_t set;
    check_ok(sigemptyset(&set), 0, "Error creating signal set");
    check_ok(sigaddset(&set, SIGUSR1), 0, "Error adding signal in set");
    check_ok(sigaddset(&set, SIGUSR2), 0, "Error adding signal in set");
```

```
43      check_ok(sigprocmask(SIG_BLOCK, &set, NULL), 0, "Error adding blocking signals in
            main!");
44
45      int id = fork();
46      check_wrong(id, -1, "Fork creating error!");
47      if (id == 0)
48      {
49          printf("[%d] It's child process\n", getpid());
50        char* arg[] = {"child", SHARED_FILE_NAME, NULL};
51          check_wrong(dup2(fileno(input), 0), -1, "Error creating duplicate file
                descriptor for input file!");
52        check_wrong(execv("child", arg), -1, "Error when starting a child for execution!"
              );
53      }
54      else
55      {
56          printf("[%d] It's parent process of %d\n", getpid(), id);
57          int sg;
58          check_ok(sigwait(&set, &sg), 0, "Error waiting signal from child process!");
59          check_ok(sg, SIGUSR2, "Error! Unknown signal from child process!");
60          while (1)
61          {
62              check_ok(kill(id, SIGUSR1), 0, "Error sending signal to child process!");
63              check_ok(sigwait(&set, &sg), 0, "Error waiting signal from child process!")
                    ;
64              check_ok(sigaddset(&set, sg), 0, "Error adding signal in set");
65              printf("%s", data);
66              for (int i = 0; i < SIZE_SHARED_FILE; ++i)
67              {
68                  data[i] = '\0';
69              }
70              if (sg == SIGUSR2)
71              {
72                  break;
73              }
74          }
75      }
76      check_ok(munmap(data, SIZE_SHARED_FILE), 0, "Error unmapping shared file in main
            process!");
77      check_wrong(shm_unlink(SHARED_FILE_NAME), -1, "Error unlink shared file in main
            process!");
78      check_ok(fclose(input), 0, "Error closing input file!");
79  }
```

Исходный код child.c:

```
1   #include <signal.h>
2   #include <sys/stat.h>
3   #include <sys/types.h>
4   #include <sys/mman.h>
```

```c
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#define check_ok(VALUE, OKVAL, MSG) if (VALUE != OKVAL) { printf("%s\n", MSG); return 1; }
#define check_wrong(VALUE, WRONGVAL, MSG) if (VALUE == WRONGVAL) { printf("%s\n", MSG); return 1; }

void itoa10(int val, char* data, int index){
    int old_index = index;
    while (val != 0)
    {
        data[index] = val % 10 + '0';
        val /= 10;
        ++index;
    }
    for (int i = old_index; i < old_index+(index-old_index)/2; ++i)
    {
        char c = data[i];
        data[i] = data[index-i+old_index-1];
        data[index-i+old_index-1] = c;
    }
}

int main(int argc, char* argv[])
{
    check_ok(argc, 2, "Wrong arguments in child process!");

    /* Open shared file */
    int sf_d = shm_open(argv[1], O_RDWR, S_IRWXU);
    check_wrong(sf_d, -1, "Error open shared data file!");
    struct stat statbuf;
    check_wrong(fstat(sf_d, &statbuf), -1, "Error getting file stat in child!");
    char* data = mmap(NULL, statbuf.st_size, PROT_READ | PROT_WRITE, MAP_SHARED, sf_d, 0);
    check_wrong(data, MAP_FAILED, "Error mapping shared data file in child process!");

    /* Working with signal set */
    sigset_t set;
    check_ok(sigemptyset(&set), 0, "Error creating signal set");
    check_ok(sigaddset(&set, SIGUSR1), 0, "Error adding signal in set");
    check_ok(sigprocmask(SIG_BLOCK, &set, NULL), 0, "Error adding blocking signals in child!");

    check_ok(kill(getppid(), SIGUSR2), 0, "Error sending signal to main process!");

    int num;
```

```
50      int index = 0;
51      char buffer[statbuf.st_size];
52      while (scanf("%d", &num) > 0)
53      {
54        int is_prime = 1;
55        if (num >= 0)
56        {
57            for (int i = 2; i * i <= num; ++i)
58            {
59                if (num % i == 0)
60                {
61                  is_prime = 0;
62                  break;
63                }
64            }
65        }
66          if (is_prime == 0)
67          {
68              int copy = num;
69            int size_num = 0;
70            while (copy != 0)
71            {
72                ++size_num;
73                copy /= 10;
74            }
75              if (index + size_num + 1 >= statbuf.st_size - 1)
76              {
77                  int sd;
78                  check_ok(sigwait(&set, &sd), 0, "Error waiting signal!");
79                  check_ok(sd, SIGUSR1, "Error! Unknown signal from main process!");
80                  check_ok(sigaddset(&set, SIGUSR1), 0, "Error adding signal in set");
81                  for (int i = 0; i < index; ++i)
82                  {
83                       data[i] = buffer[i];
84                       buffer[i] = '\0';
85                  }
86                  kill(getppid(), SIGUSR1);
87                  index = 0;
88              }
89              itoa10(num, buffer, index);
90              index += size_num;
91              buffer[index] = '\n';
92              ++index;
93          }
94          else
95          {
96              int sd;
97              check_ok(sigwait(&set, &sd), 0, "Error waiting signal!");
98              check_ok(sd, SIGUSR1, "Error! Unknown signal from main process!");
```

```
 99            for (int i = 0; i < index; ++i)
100            {
101                data[i] = buffer[i];
102                buffer[i] = '\0';
103            }
104            kill(getppid(), SIGUSR2);
105            break;
106        }
107    }
108    check_ok(munmap(data, statbuf.st_size), 0, "Error unmapping shared file in child
           process!");
109 }
```

4 Тесты и протокол работы

```
dmitry@dmitry-VirtualBox:~/OS_labs/Lab4$ gcc child.c -o child -lpthread -lrt
dmitry@dmitry-VirtualBox:~/OS_labs/Lab4$ gcc main.c -o main -lpthread -lrt
dmitry@dmitry-VirtualBox:~/OS_labs/Lab4$ cat test1.txt
20
10
8
8
8
10
20
33
33
33
33
40
33
2
-1
dmitry@dmitry-VirtualBox:~/OS_labs/Lab4$ ./main
Input file name: test1.txt
[4054] It's parent process of 4055
[4055] It's child process
20
10
8
8
8
10
20
33
33
33
33
40
33
dmitry@dmitry-VirtualBox:~/OS_labs/Lab4$ cat test2.txt
4
1000
333
```

```
3333
1888
1888
1888
1888
188
-5
10
2
20
dmitry@dmitry-VirtualBox:~/OS_labs/Lab4$ ./main
Input file name: test2.txt
[4058] It's parent process of 4059
[4059] It's child process
4
1000
333
3333
1888
1888
1888
1888
188
dmitry@dmitry-VirtualBox:~/OS_labs/Lab4$
```

5 Strace

```
dmitry@dmitry-VirtualBox:~/Work_place/OS_labs/Lab4$ strace -f ./main
execve("./main",["./main"],0x7fff0e82d7c8 /* 49 vars */) = 0
brk(NULL)                                = 0x564086053000
arch_prctl(0x3001 /* ARCH_??? */,0x7ffff8520260) = -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=67205,...) = 0
mmap(NULL,67205,PROT_READ,MAP_PRIVATE,3,0) = 0x7f7421d30000
close(3)                                 = 0
openat(AT_FDCWD,"/lib/x86_64-linux-gnu/libpthread.so.0",O_RDONLY|O_CLOEXEC)
= 3
read(3,"77ELF>2001
...,832) = 832
pread64(3,"4GNUO0574364B216442406@ 261327o" ...,68,824) = 68
fstat(3,st_mode=S_IFREG|0755,st_size=157224,...) = 0
mmap(NULL,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS,-1,0)
= 0x7f7421d2e000
pread64(3,"4GNUO0574364B216442406@
61327o" ...,68,824) = 68
mmap(NULL,140408,PROT_READ,MAP_PRIVATE|MAP_DENYWRITE,3,0) = 0x7f7421d0b000
mmap(0x7f7421d12000,69632,PROT_READ|PROT_EXEC,MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE,3,0x7000) = 0x7f7421d12000
mmap(0x7f7421d23000,20480,PROT_READ,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3,0x18000) = 0x7f7421d23000
mmap(0x7f7421d28000,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE,3,0x1c000) = 0x7f7421d28000
mmap(0x7f7421d2a000,13432,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|
MAP_ANONYMOUS,-1,0) = 0x7f7421d2a000
close(3)                                 = 0
openat(AT_FDCWD,"/lib/x86_64-linux-gnu/librt.so.1",O_RDONLY|O_CLOEXEC) = 3
read(3,"77ELF>7"...,832) = 832
fstat(3,st_mode=S_IFREG|0644,st_size=40040,...) = 0
mmap(NULL,44000,PROT_READ,MAP_PRIVATE|MAP_DENYWRITE,3,0) = 0x7f7421d00000
munmap(0x7f7421d30000,67205)           = 0
set_tid_address(0x7f7421b0ba10)         = 4084
set_robust_list(0x7f7421b0ba20,24)     = 0
rt_sigaction(SIGRTMIN,sa_handler=0x7f7421d12bf0,sa_mask=[],
sa_flags=SA_RESTORER|SA_SIGINFO,sa_restorer=0x7f7421d203c0,NULL,8) = 0
```

```
rt_sigaction(SIGRT_1,sa_handler=0x7f7421d12c90,sa_mask=[],
sa_flags=SA_RESTORER|SA_RESTART| SA_SIGINFO,sa_restorer=0x7f7421d203c0,
NULL,8) = 0
rt_sigprocmask(SIG_UNBLOCK,[RTMIN RT_1],NULL,8) = 0
prlimit64(0,RLIMIT_STACK,NULL,rlim_cur=8192*1024,rlim_max=RLIM64_INFINITY)
= 0
fstat(1,st_mode=S_IFCHR|0620,st_rdev=makedev(0x88,0),...) = 0
brk(NULL)                                = 0x564086053000
brk(0x564086074000)                      = 0x564086074000
fstat(0,st_mode=S_IFCHR|0620,st_rdev=makedev(0x88,0),...) = 0
write(1,"Input file name: ",17Input file name: )       = 17
read(0,input.txt
"input.txt",1024)            = 10
openat(AT_FDCWD,"input.txt",O_RDONLY) = 3
statfs("/dev/shm/",f_type=TMPFS_MAGIC,f_bsize=4096,f_blocks=393573,f_bfree=393573,
f_bavail=393573,f_files=393573,f_ffree=393572,f_fsid=val=[0,0],f_namelen=255,
f_frsize=4096,f_flags=ST_VALID|ST_NOSUID|ST_NODEV) = 0
futex(0x7f7421d2d390,FUTEX_WAKE_PRIVATE,2147483647) = 0
openat(AT_FDCWD,"/dev/shm/pipe",O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC,0700) =
4
ftruncate(4,12)                          = 0
mmap(NULL,12,PROT_READ|PROT_WRITE,MAP_SHARED,4,0) = 0x7f7421d6d000
rt_sigprocmask(SIG_BLOCK,[USR1 USR2],NULL,8) = 0
clone(child_stack=NULL,flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7f7421b0ba10) = 4085
getpid()                                 = 4084
write(1,"[4084] It's parent process of 40"...,35[4084] It's parent process
of 4085
) = 35
rt_sigtimedwait([USR1 USR2],strace: Process 4085 attached
<unfinished ...>
[pid  4085] set_robust_list(0x7f7421b0ba20,24) = 0
[pid  4085] getpid()                     = 4085
[pid  4085] write(1,"[4085] It's child process",26[4085] It's child process
) = 26
[pid  4085] dup2(3,0)                    = 0
[pid  4085] execve("child",["child","pipe"],0x7ffff8520348 /* 49 vars */) =
0
[pid  4085] brk(NULL)                    = 0x55ffe637f000
[pid  4085] arch_prctl(0x3001 /* ARCH_??? */,0x7ffd28d29800)
= -1 EINVAL (Недопустимый аргумент)
```

```
[pid  4085] access("/etc/ld.so.preload",R_OK)
= -1 ENOENT (Нет такого файла или каталога)
[pid  4085] openat(AT_FDCWD,"/etc/ld.so.cache",O_RDONLY|O_CLOEXEC) = 4
[pid  4085] fstat(4,st_mode=S_IFREG|0644,st_size=67205,...) = 0
[pid  4085] mmap(NULL,67205,PROT_READ,MAP_PRIVATE,4,0) = 0x7fee42ccc000
[pid  4085] close(4)                    = 0
[pid  4085] openat(AT_FDCWD,"/lib/x86_64-linux-gnu/libpthread.so.0",
O_RDONLY|O_CLOEXEC) = 4
[pid  4085] read(4,"77ELF>2001"
...,832) = 832
[pid  4085] pread64(4,"4GNUO0574364B216442406@ 261327o"...,68,824) = 68
[pid  4085] fstat(4,st_mode=S_IFREG|0755,st_size=157224,...) = 0
[pid  4085] mmap(NULL,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS,-1,0)
= 0x7fee42cca000
[pid  4085] pread64(4,"4GNUO0574364B216442406@ 261327o"...,68,824) = 68
[pid  4085] mmap(NULL,140408,PROT_READ,MAP_PRIVATE|MAP_DENYWRITE,4,0)
= 0x7fee42ca7000
[pid  4085] mmap(0x7fee42cae000,69632,PROT_READ|PROT_EXEC,MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE,4,0x7000) = 0x7fee42cae000
[pid  4085] mmap(0x7fee42cbf000,20480,PROT_READ,MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE,4,0x18000) = 0x7fee42cbf000
[pid  4085] mmap(0x7fee42cc4000,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE,4,0x1c000) = 0x7fee42cc4000
[pid  4085] mmap(0x7fee42cc6000,13432,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|
MAP_ANONYMOUS,-1,0) = 0x7fee42cc6000
[pid  4085] close(4)                    = 0
[pid  4085] openat(AT_FDCWD,"/lib/x86_64-linux-gnu/librt.so.1",O_RDONLY|
O_CLOEXEC) = 4
[pid  4085] read(4,"77ELF>7"...,
832) = 832
[pid  4085] fstat(4,st_mode=S_IFREG|0644,st_size=40040,...) = 0
[pid  4085] mmap(NULL,44000,PROT_READ,MAP_PRIVATE|MAP_DENYWRITE,4,0)
= 0x7fee42c9c000
[pid  4085] mprotect(0x7fee42c9f000,24576,PROT_NONE) = 0
[pid  4085] mmap(0x7fee42c9f000,16384,PROT_READ|PROT_EXEC,MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE,4,0x3000) = 0x7fee42c9f000
[pid  4085] mmap(0x7fee42ca3000,4096,PROT_READ,MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE,4,0x7000) = 0x7fee42ca3000
[pid  4085] mmap(0x7fee42ca5000,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE,4,0x8000) = 0x7fee42ca5000
[pid  4085] close(4)                    = 0
```

```
[pid  4085] close(4)                    = 0
[pid  4085] mmap(NULL,12288,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS,
-1,0) = 0x7fee42aa7000
[pid  4085] arch_prctl(ARCH_SET_FS,0x7fee42aa7740) = 0
[pid  4085] mprotect(0x7fee42c92000,12288,PROT_READ) = 0
[pid  4085] mprotect(0x7fee42cc4000,4096,PROT_READ) = 0
[pid  4085] mprotect(0x7fee42ca5000,4096,PROT_READ) = 0
[pid  4085] mprotect(0x55ffe5ba0000,4096,PROT_READ) = 0
[pid  4085] mprotect(0x7fee42d0a000,4096,PROT_READ) = 0
[pid  4085] munmap(0x7fee42ccc000,67205) = 0
[pid  4085] set_tid_address(0x7fee42aa7a10) = 4085
[pid  4085] set_robust_list(0x7fee42aa7a20,24) = 0
[pid  4085] rt_sigaction(SIGRTMIN,sa_handler=0x7fee42caebf0,sa_mask=[],
sa_flags=SA_RESTORER| SA_SIGINFO,sa_restorer=0x7fee42cbc3c0,NULL,8) = 0
[pid  4085] rt_sigaction(SIGRT_1,sa_handler=0x7fee42caec90,sa_mask=[],
sa_flags=SA_RESTORER| SA_RESTART|SA_SIGINFO,sa_restorer=0x7fee42cbc3c0,NULL,8)
= 0
[pid  4085] rt_sigprocmask(SIG_UNBLOCK,[RTMIN RT_1],NULL,8) = 0
[pid  4085] prlimit64(0,RLIMIT_STACK,NULL,rlim_cur=8192*1024,
rlim_max=RLIM64_INFINITY) = 0
[pid  4085] statfs("/dev/shm/",f_type=TMPFS_MAGIC,f_bsize=4096,
f_blocks=393573,f_bfree=393572,f_bavail=393572,f_files=393573,f_ffree=393571,
f_fsid=val=[0,0],f_namelen=255,f_frsize=4096,f_flags=ST_VALID|
ST_NOSUID|ST_NODEV) = 0
[pid  4085] futex(0x7fee42cc9390,FUTEX_WAKE_PRIVATE,2147483647) = 0
[pid  4085] openat(AT_FDCWD,"/dev/shm/pipe",O_RDWR|O_NOFOLLOW|O_CLOEXEC) =
4
[pid  4085] fstat(4,st_mode=S_IFREG|0700,st_size=12,...) = 0
[pid  4085] mmap(NULL,12,PROT_READ|PROT_WRITE,MAP_SHARED,4,0) = 0x7fee42d09000
[pid  4085] rt_sigprocmask(SIG_BLOCK,[USR1],NULL,8) = 0
[pid  4085] getppid()                   = 4084
[pid  4085] kill(4084,SIGUSR2 <unfinished ...>
[pid  4084] <... rt_sigtimedwait resumed>si_signo=SIGUSR2,si_code=SI_USER,
si_pid=4085,si_uid=1000,NULL,8) = 12 (SIGUSR2)
[pid  4085] <... kill resumed>)         = 0
[pid  4084] kill(4085,SIGUSR1 <unfinished ...>
[pid  4085] fstat(0, <unfinished ...>
[pid  4084] <... kill resumed>)         = 0
[pid  4085] <... fstat resumed>st_mode=S_IFREG|0664,st_size=161,...) = 0
[pid  4084] rt_sigtimedwait([USR1 USR2], <unfinished ...>
[pid  4085] brk(NULL)                   = 0x55ffe637f000
```

```
[pid  4085] brk(0x55ffe63a0000)         = 0x55ffe63a0000
[pid  4085] read(0,"201088810203333333340"...,4096)
= 161
[pid  4085] rt_sigtimedwait([USR1],si_signo=SIGUSR1,si_code=SI_USER,
si_pid=4084,si_uid=1000,NULL,8) = 10 (SIGUSR1)
[pid  4085] getppid()                   = 4084
[pid  4085] kill(4084,SIGUSR1 <unfinished ...>
[pid  4084] <... rt_sigtimedwait resumed>si_signo=SIGUSR1,
si_code=SI_USER,si_pid=4085,si_uid=1000,NULL,8) = 10 (SIGUSR1)
[pid  4085] <... kill resumed>)         = 0
[pid  4084] write(1,"201088",10
20
10
8
8
<unfinished ...>
[pid  4085] rt_sigtimedwait([USR1], <unfinished ...>
[pid  4084] <... write resumed>)        = 6
[pid  4084] kill(4085,SIGUSR1)         = 0
[pid  4085] <... rt_sigtimedwait resumed>si_signo=SIGUSR1,si_code=SI_USER,
si_pid=4084,si_uid=1000,NULL,8) = 10 (SIGUSR1)
[pid  4084] rt_sigtimedwait([USR1 USR2], <unfinished ...>
[pid  4085] getppid()                   = 4084
[pid  4085] kill(4084,SIGUSR1 <unfinished ...>
[pid  4084] <... rt_sigtimedwait resumed>si_signo=SIGUSR1,si_code=SI_USER,
si_pid=4085,si_uid=1000,NULL,8) = 10 (SIGUSR1)
[pid  4085] <... kill resumed>)         = 0
[pid  4084] write(1,"888888",7
888888
<unfinished ...>
[pid  4085] rt_sigtimedwait([USR1], <unfinished ...>
[pid  4084] <... write resumed>)        = 7
[pid  4084] kill(4085,SIGUSR1)         = 0
[pid  4085] <... rt_sigtimedwait resumed>si_signo=SIGUSR1,si_code=SI_USER,
si_pid=4084,si_uid=1000,NULL,8) = 10 (SIGUSR1)
[pid  4084] rt_sigtimedwait([USR1 USR2], <unfinished ...>
[pid  4085] getppid()                   = 4084
[pid  4085] kill(4084,SIGUSR2 <unfinished ...>
[pid  4084] <... rt_sigtimedwait resumed>si_signo=SIGUSR2,si_code=SI_USER,
si_pid=4085,si_uid=1000,NULL,8) = 12 (SIGUSR2)
[pid  4085] <... kill resumed>)         = 0
```

```
[pid  4084] write(1,"88888",7
8888
8
<unfinished ...>
[pid  4085] munmap(0x7fee42d09000,12 <unfinished ...>
[pid  4084] <... write resumed>)        = 7
[pid  4085] <... munmap resumed>)       = 0
[pid  4084] munmap(0x7f7421d6d000,12 <unfinished ...>
[pid  4085] lseek(0,-7,SEEK_CUR <unfinished ...>
[pid  4084] <... munmap resumed>)       = 0
[pid  4085] <... lseek resumed>)        = 154
[pid  4084] unlink("/dev/shm/pipe" <unfinished ...>
[pid  4085] exit_group(0 <unfinished ...>
[pid  4084] <... unlink resumed>)       = 0
[pid  4085] <... exit_group resumed>)   = ?
[pid  4084] close(3 <unfinished ...>
[pid  4085] +++ exited with 0 +++
<... close resumed>)                    = 0
---SIGCHLD si_signo=SIGCHLD,si_code=CLD_EXITED,si_pid=4085,si_uid=1000,
si_status=0,si_utime=0,si_stime=0 ---
exit_group(0)                           = ?
+++ exited with 0 +++
dmitry@dmitry-VirtualBox:~/Work_place/OS_labs/Lab4$
```

Как видно, сперва вызывается функция execv(), которая запускает файл программы на исполнение. Печать и вывод в консоль с использованием функций scanf и printf реализуется соответственно с помощью системных вызовов read и write, где в аргументах указывается буфер, откуда читаются или записываются данные, их размер в байтах, а также дескриптор файла, откуда читаются или записываются данные (0 - стандартный поток ввода, 1 - стандартный поток вывода).

При открытии входного файла на чтение, а также при создании общего файла происходит вызов функции openat, где указывается имя файла (с добавлением информации о том, что он находится в текущем рабочем каталоге – ключ AT_FDCWD), режим работы, и возвращается дескриптор файла.

Создание дочернего процесса происходит с помощью системного вызова clone, далее происходит переопределение файловых дескрипторов стандартного потока ввода и файла с входными данными с помощью dup2, затем заменяется образ процесса вызовом execv с указанием имени программы child и её аргументом с именем общего файла.

Переопределение реакции программы на сигналы происходит с помощью rt_sigaction

с указанием сигнала, действия при получении этого сигнала, а также возможное сохранение старого действия для сигнала. Далее происходит вызов rt_sigprocmask, который объединяет полученные набор сигналов с набором блокирующих сигналов текущего процесса. При ожидании получения сигналов вызывается rt_sigtimedwait, в котором указывается набор получаемых сигналов, а также место, куда будет записана информация о полученном сигнала (возвращаемое значение функции соответствует номеру полученного сигнала). В свою очередь отправление сигналов процессом осуществляется путем системного вызова kill, где указывается номер процесса-получателя, а также сам сигнал.

6 Выводы

В результате выполнения лабораторной работы №4 я познакомился с использованием технологии «File mapping» для обеспечения обмена данными между процессами, которая является довольно эффективной, поскольку позволяет избавиться от операции копирования блоков файлов при их чтении/записи из кэша в буфер, отображая их прямо из кэша в память.

Также я научился использовать сигналы для взаимодействия процессов, однако в ходе налаживания такой связи я столкнулся с рядом трудностей. Например, необходимо было правильно предусмотреть место переопределения набора блокирующих сигналов процесса, чтобы в случае получения сигнала не произошло убийство процесса, поскольку по умолчанию пользовательские сигналы как раз завершают процесс, их получивший. Как мне кажется, сигналы будут недостаточными при создании больших многопоточных программ, поскольку в POSIX было предусмотрено только возможное переопределение двух сигналов под свои нужды, чего явно мало при реальной работе.

Список литературы

[1] Таненбаум Э., Босх Х. *Современные операционные системы. – 4-е изд.* – СПб.: Издательский дом «Питер», 2018. – С. 301-380

[2] *Linux Man Pages*
URL: `http://ru.manpages.org` (дата обращения: 27.11.2020).