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: М8О-211Б-23

Студент: Воробьев Г. Я.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 18.11.24

Москва, 2024

Постановка задачи

Вариант 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. Количество чисел может быть произвольным.

Общий метод и алгоритм решения

Использованные системные вызовы:

- pid_t fork(void); – создает дочерний процесс.
- int pipe(int* fd); – создает pipe, однонаправленный канал связи между процессами
- ssize_t write(int fd, const void* buf, ssize_t count); – пишет count байтов из буфера в файл, на который ссылается файловый дескриптор fd
- ssize_t read(int fd, void* buf, size_t count) – пытается прочитать count байтов из файлового дескриптора fd в буфер buff
- pid_t getpid(void); - получить pid текущего или родительского процесса
- int open(const char* pathname, int flags, mode _ t mode); – открыть файл с указанными флагами или создать, если указаны специальные флаги, возвращает файловый дескриптор
- int close(int fd); - закрывает файловый дескриптор
- int execv(const char* pathname, char* const argv[]); – заменяет образ текущего процесса на новый образ, создается новый стек, кучу и сегменты данных
- pid_t waitpid (pid_t pid, int* stat_loc, int options); – ждет пока процесс с pid завершится и получается код выхода
- int dup2(int oldfd, int newfd) - переназначение файлового дескриптора
- void exit(int status) - завершения выполнения процесса и возвращение статуса

Программа получает из стандартного ввода имя файла, создает pipe и создает дочерний процесс с помощью fork. Перед запуском кода дочернего процесса стандартный поток ввода дочернего процесса переопределяется открытым файлом через dup2. Далее в дочерний процесс через аргументы передается имя файла с кодом дочернего процесса, который запускается через execv. Данные операции повторяются еще раз для второго дочернего процесса. Дочерний процесс читает команды из стандартного потока ввода и производит проверку чисел, записанных в файле 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. Родительский процесс читает из pipe и прочитанное выводит в свой стандартный поток вывода. Дочерние процессы завершаются, а

родительский процесс ждет их заверения. Все процессы закрывают в конце закрывают открытые файлы и pipe.

Код программы

main.c

```c
#include "../inc/parent.h"



void get_input(char *input, size_t size);



int main() {

   char progpath[4096];

   get_program_path(progpath, sizeof(progpath));



   char file[4096];



   {

       char msg[32];

       const int32_t length = snprintf(msg, sizeof(msg), "Print file name:\n");

       write(STDOUT_FILENO, msg, length);

   }



   get_input(file, sizeof(file));



   int channel[2];

   create_pipe(channel);



   int file_fd = open(file, O_RDONLY);

   if (file_fd == -1) {

       const char msg[] = "error: failed to open requested file\n";

       write(STDERR_FILENO, msg, sizeof(msg));

       exit(EXIT_FAILURE);

   }
```

```
    const pid_t child = fork();

    switch (child) {
    case -1:
        {
            const char msg[] = "error: failed to spawn new process\n";
            write(STDERR_FILENO, msg, sizeof(msg));
            exit(EXIT_FAILURE);
        }
        break;

    case 0:
        handle_child_process(channel, progpath, file_fd);
        break;

    default:
        handle_parent_process(channel, child);
        break;
    }

    close(file_fd);
    return 0;
}

void get_input(char *input, size_t size)
{
    if (fgets(input, size, stdin) == NULL) {
        char msg[32];
        const int32_t length = snprintf(msg, sizeof(msg), "error: failed to read
input\n");
```

```
        write(STDERR_FILENO, msg, sizeof(msg));

        exit(EXIT_FAILURE);

    }



    size_t len = strlen(input);

    if (len > 0 && input[len - 1] == '\n') {

        input[len - 1] = '\0';

    }

}
```

core.c

```
#include "../inc/parent.h"



void get_program_path(char *progpath, size_t size) {

    uint32_t path_size = (uint32_t)size;

    if (_NSGetExecutablePath(progpath, &path_size) != 0) {

        const char msg[] = "error: failed to read full program path\n";

        write(STDERR_FILENO, msg, sizeof(msg));

        exit(EXIT_FAILURE);

    }



    char *last_slash = strrchr(progpath, '/');

    if (last_slash != NULL) {

        *last_slash = '\0';

    }

}



void create_pipe(int channel[2]) {

    if (pipe(channel) == -1) {

        const char msg[] = "error: failed to create pipe\n";
```

```
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }
}

void handle_child_process(int channel[2], char *progpath, int file_fd) {
    pid_t pid = getpid();
    {
        char msg[64];
        const int32_t length = snprintf(msg, sizeof(msg), "%d: I'm a child\n", pid);
        write(STDOUT_FILENO, msg, length);
    }

    dup2(file_fd, STDIN_FILENO);
    close(file_fd);

    dup2(channel[STDOUT_FILENO], STDOUT_FILENO);
    close(channel[STDOUT_FILENO]);

    char path[4096];
    snprintf(path, sizeof(path) - 1, "%s/%s", progpath, CLIENT_PROGRAM_NAME);

    char* const args[] = {CLIENT_PROGRAM_NAME, NULL};
    int32_t status = execv(path, args);

    if (status == -1) {
        const char msg[] = "error: failed to exec into new executable image\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }
}
```

```
void handle_parent_process(int channel[2], pid_t child) {
   pid_t pid = getpid();
   {
       char msg[64];
       const int32_t length = snprintf(msg, sizeof(msg), "%d: I'm a parent, my child has
PID %d\n", pid, child);
       write(STDOUT_FILENO, msg, length);
   }

   close(channel[1]);

   char buf[256];
   ssize_t bytes;
   while ((bytes = read(channel[0], buf, sizeof(buf))) > 0)
       write(STDOUT_FILENO, buf, bytes);

   close(channel[0]);

   int child_status;
   wait(&child_status);

   if (child_status != EXIT_SUCCESS) {
       const char msg[] = "error: child exited with error\n";
       write(STDERR_FILENO, msg, sizeof(msg));
       exit(child_status);
   }
}
```

child.c

```c
#include "../inc/child.h"



int main() {

   char buf[256];

   int num;

   ssize_t bytes_read;



   while ((bytes_read = read(STDIN_FILENO, buf, sizeof(buf))) > 0) {

       char *ptr = buf;

       while (ptr < buf + bytes_read) {

           char *endptr;

           num = strtol(ptr, &endptr, 10);



           if (ptr == endptr) {

               while (ptr < buf + bytes_read && *ptr != '\n') {

                   ptr++;

               }

               ptr++;

               continue;

           }



           if (num < 0 || is_prime(num)) {

               exit(EXIT_SUCCESS);

           }



           if (!is_prime(num)) {

               char output[32];

               int len = snprintf(output, sizeof(output), "%d\n", num);

               write(STDOUT_FILENO, output, len);

           }
```

```
            while (ptr < buf + bytes_read && *ptr != '\n') {
                ptr++;
            }
            ptr++;
        }
    }

    return 0;
}

int is_prime(int num) {
    if (num <= 1)
        return 0;

    if (num <= 3)
        return 1;

    if (num % 2 == 0 || num % 3 == 0)
        return 0;

    for (int i = 5; i * i <= num; i += 6)
        if (num % i == 0 || num % (i + 2) == 0)
            return 0;

    return 1;
}
```

Протокол работы программы

Тестирование:

```
glebvorobev@Glebs-MacBook-Air LAB_1 % ./parent
Print file name:
test.txt
1764: I'm a parent, my child has PID 1767
1767: I'm a child
8
4
6
glebvorobev@Glebs-MacBook-Air LAB_1 % cat test.txt
8
4
6
3
glebvorobev@Glebs-MacBook-Air LAB_1 % ./parent
Print file name:
test.txt
1840: I'm a parent, my child has PID 1841
1841: I'm a child
8
4
6
glebvorobev@Glebs-MacBook-Air LAB_1 % cat test.txt
8
4
6
-4
```

Dtruss:

```
glebvorobev@Glebs-MacBook-Air LAB_1 % sudo dtruss ./parent
Password:
SYSCALL(args)           = return
munmap(0x1011C8000, 0x84000)            = 0 0
munmap(0x10124C000, 0x8000)             = 0 0
munmap(0x101254000, 0x4000)             = 0 0
munmap(0x101258000, 0x4000)             = 0 0
Print file name:
munmap(0x10125C000, 0x48000)            = 0 0
munmap(0x1012A4000, 0x4C000)            = 0 0
crossarch_trap(0x0, 0x0, 0x0)           = -1 Err#45
open(".\0", 0x100000, 0x0)              = 3 0
fcntl(0x3, 0x32, 0x16EF53188)           = 0 0
close(0x3)              = 0 0
fsgetpath(0x16EF53198, 0x400, 0x16EF53178)              = 60 0
fsgetpath(0x16EF531A8, 0x400, 0x16EF53188)              = 14 0
csrctl(0x0, 0x16EF535AC, 0x4)           = -1 Err#1
__mac_syscall(0x187573C12, 0x2, 0x16EF534F0)            = 0 0
```

```
csrctl(0x0, 0x16EF5359C, 0x4)            = -1 Err#1
__mac_syscall(0x187570A45, 0x5A, 0x16EF53530)            = 0 0
sysctl([unknown, 3, 0, 0, 0, 0] (2), 0x16EF52A98, 0x16EF52A90, 0x187572738, 0xD)
               = 0 0
sysctl([CTL_KERN, 150, 0, 0, 0, 0] (2), 0x16EF52B48, 0x16EF52B40, 0x0, 0x0)
            = 0 0
open("/\0", 0x20100000, 0x0)             = 3 0
openat(0x3, "System/Cryptexes/OS\0", 0x100000, 0x0)              = 4 0
dup(0x4, 0x0, 0x0)               = 5 0
fstatat64(0x4, 0x16EF52621, 0x16EF52590)                 = 0 0
openat(0x4, "System/Library/dyld/\0", 0x100000, 0x0)             = 6 0
fcntl(0x6, 0x32, 0x16EF52620)            = 0 0
dup(0x6, 0x0, 0x0)               = 7 0
dup(0x5, 0x0, 0x0)               = 8 0
close(0x3)               = 0 0
close(0x5)               = 0 0
close(0x4)               = 0 0
close(0x6)               = 0 0
__mac_syscall(0x187573C12, 0x2, 0x16EF53010)             = 0 0
shared_region_check_np(0x16EF52C30, 0x0, 0x0)            = 0 0
fsgetpath(0x16EF531B0, 0x400, 0x16EF530D8)               = 82 0
fcntl(0x8, 0x32, 0x16EF531B0)            = 0 0
close(0x8)               = 0 0
close(0x7)               = 0 0
getfsstat64(0x0, 0x0, 0x2)               = 10 0
getfsstat64(0x100EA8050, 0x54B0, 0x2)            = 10 0
getattrlist("/\0", 0x16EF530F0, 0x16EF53060)             = 0 0
stat64("/System/Volumes/Preboot/Cryptexes/OS/System/Library/dyld/dyld_shared_cache_arm
64e\0", 0x16EF53450, 0x0)          = 0 0
dtrace: error on enabled probe ID 1696 (ID 845: syscall::stat64:return): invalid
address (0x0) in action #11 at DIF offset 12
stat64("/Users/glebvorobev/Documents/Projects/OS/build/LAB_1/parent\0", 0x16EF52900,
 0x0)               = 0 0
open("/Users/glebvorobev/Documents/Projects/OS/build/LAB_1/parent\0", 0x0, 0x0)
      = 3 0
mmap(0x0, 0x8A98, 0x1, 0x40002, 0x3, 0x0)                = 0x100EA8000 0
fcntl(0x3, 0x32, 0x16EF52A18)            = 0 0
close(0x3)               = 0 0
munmap(0x100EA8000, 0x8A98)              = 0 0
stat64("/Users/glebvorobev/Documents/Projects/OS/build/LAB_1/parent\0", 0x16EF52E70,
 0x0)               = 0 0
stat64("/usr/lib/libSystem.B.dylib\0", 0x16EF51DC0, 0x0)                 = -1 Err#2
stat64("/System/Volumes/Preboot/Cryptexes/OS/usr/lib/libSystem.B.dylib\0",
0x16EF51D70, 0x0)             = -1 Err#2
open("/Users/glebvorobev/Documents/Projects/OS/build/LAB_1/parent\0", 0x0, 0x0)
       = 3 0
__mac_syscall(0x187573C12, 0x2, 0x16EF504C0)             = 0 0
map_with_linking_np(0x16EF50340, 0x1, 0x16EF50370)               = 0 0
close(0x3)               = 0 0
```

```
mprotect(0x100EA0000, 0x4000, 0x1)              = 0 0
open("/dev/dtracehelper\0", 0x2, 0x0)           = 3 0
ioctl(0x3, 0x80086804, 0x16EF4FA78)             = 0 0
close(0x3)              = 0 0
shared_region_check_np(0xFFFFFFFFFFFFFFFF, 0x0, 0x0)            = 0 0
access("/AppleInternal/XBS/.isChrooted\0", 0x0, 0x0)            = -1 Err#2
bsdthread_register(0x1878760F4, 0x1878760E8, 0x4000)            = 1073746399 0
getpid(0x0, 0x0, 0x0)           = 1473 0
shm_open(0x18770DF41, 0x0, 0xFFFFFFFF878B4000)          = 3 0
fstat64(0x3, 0x16EF500F0, 0x0)          = 0 0
mmap(0x0, 0x8000, 0x1, 0x40001, 0x3, 0x0)               = 0x100EB0000 0
close(0x3)              = 0 0
csops(0x5C1, 0x0, 0x16EF5022C)          = 0 0
ioctl(0x2, 0x4004667A, 0x16EF5019C)             = 0 0
mprotect(0x100EC0000, 0x4000, 0x0)              = 0 0
mprotect(0x100ECC000, 0x4000, 0x0)              = 0 0
mprotect(0x100ED0000, 0x4000, 0x0)              = 0 0
mprotect(0x100EDC000, 0x4000, 0x0)              = 0 0
mprotect(0x100EE0000, 0x4000, 0x0)              = 0 0
mprotect(0x100EEC000, 0x4000, 0x0)              = 0 0
mprotect(0x100EB8000, 0xC8, 0x1)                = 0 0
mprotect(0x100EB8000, 0xC8, 0x3)                = 0 0
mprotect(0x100EB8000, 0xC8, 0x1)                = 0 0
mprotect(0x100EF0000, 0x4000, 0x1)              = 0 0
mprotect(0x100EF4000, 0xC8, 0x1)                = 0 0
mprotect(0x100EF4000, 0xC8, 0x3)                = 0 0
mprotect(0x100EF4000, 0xC8, 0x1)                = 0 0
mprotect(0x100EB8000, 0xC8, 0x3)                = 0 0
mprotect(0x100EB8000, 0xC8, 0x1)                = 0 0
mprotect(0x100EF0000, 0x4000, 0x3)              = 0 0
mprotect(0x100EF0000, 0x4000, 0x1)              = 0 0
issetugid(0x0, 0x0, 0x0)                = 0 0
getentropy(0x16EF4F7F8, 0x20, 0x0)              = 0 0
getattrlist("/Users/glebvorobev/Documents/Projects/OS/build/LAB_1/parent\0",
0x16EF50090, 0x16EF500AC)           = 0 0
access("/Users/glebvorobev/Documents/Projects/OS/build/LAB_1\0", 0x4, 0x0)
               = 0 0
open("/Users/glebvorobev/Documents/Projects/OS/build/LAB_1\0", 0x0, 0x0)
                 = 3 0
fstat64(0x3, 0x13CE04430, 0x0)          = 0 0
csrctl(0x0, 0x16EF5027C, 0x4)           = 0 0
fcntl(0x3, 0x32, 0x16EF4FF78)           = 0 0
close(0x3)              = 0 0
open("/Users/glebvorobev/Documents/Projects/OS/build/LAB_1/Info.plist\0", 0x0, 0x0)
            = -1 Err#2
proc_info(0x2, 0x5C1, 0xD)              = 64 0
csops_audittoken(0x5C1, 0x10, 0x16EF50300)              = 0 0
sysctl([unknown, 3, 0, 0, 0, 0] (2), 0x16EF50658, 0x16EF50650, 0x18AFA0D3A, 0x15)
               = 0 0
```

```
sysctl([CTL_KERN, 148, 0, 0, 0, 0] (2), 0x16EF506E8, 0x16EF506E0, 0x0, 0x0)
        = 0 0
write(0x1, "Print file name:\n\0", 0x11)                = 17 0
fstat64(0x0, 0x16EF614F0, 0x0)          = 0 0
ioctl(0x0, 0x4004667A, 0x16EF6153C)             = 0 0
test.txt
1473: I'm a parent, my child has PID 1478
1478: I'm a child
8
4
6
read_nocancel(0x0, "test.txt\n\0", 0x1000)              = 9 0
pipe(0x0, 0x0, 0x0)             = 3 0
open("test.txt\0", 0x0, 0x0)            = 5 0
fork()          = 1478 0
write(0x1, "1473: I'm a parent, my child has PID 1478\n\0", 0x2A)                = 42 0
close(0x4)              = 0 0
read(0x3, "8\n4\n6\n\0", 0x1000)                = 6 0
write(0x1, "8\n4\n6\n\0", 0x6)          = 6 0
read(0x3, "\0", 0x1000)         = 0 0
close(0x3)              = 0 0
wait4(0xFFFFFFFFFFFFFFFF, 0x16EF6063C, 0x0)             = 1478 0
close(0x5)              = 0 0
```

Вывод

В ходе проделанной работы я научился работать с процессами. А именно налаживать связь между ними через каналы, перенаправлять стандартный поток ввода/вывода и т. д. С неразрешимыми трудностями не столкнулся.