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: М8О-213Б-23

Студент: Германенко М. И.

Преподаватель: Бахарев В.Д.

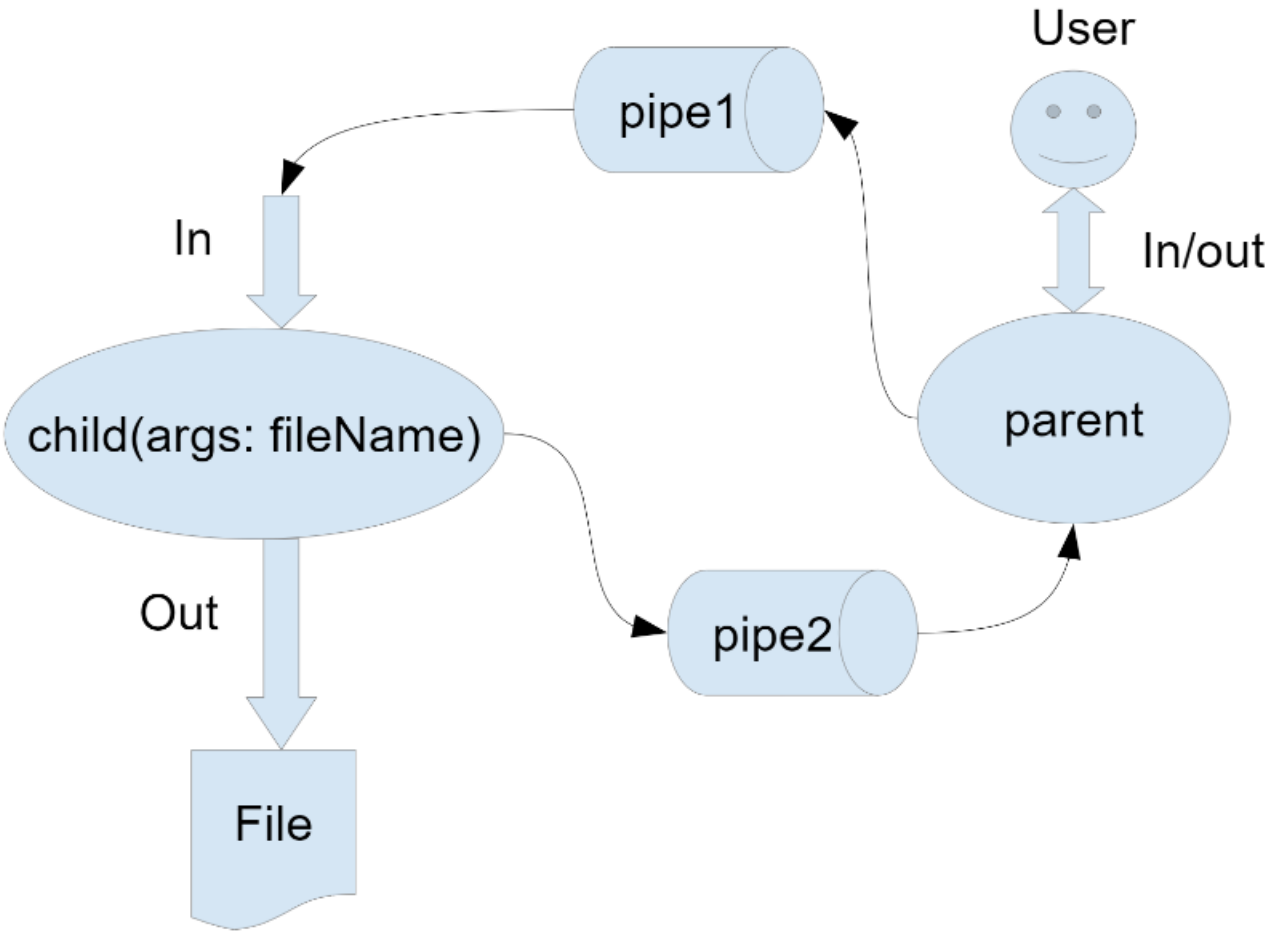
Оценка: _____

Дата: 25.11.24

Москва, 2024

Постановка задачи

Вариант 5.



Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса пишет имя файла, которое будет передано при создании дочернего процесса. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс передает команды пользователя через pipe1, который связан с стандартным входным потоком дочернего процесса. Дочерний процесс принеобходимости передает данные в родительский процесс через pipe2. Результаты своей работы дочерний процесс пишет в созданный им файл. Допускается просто открыть файл и писать туда, не перенаправляя стандартный поток вывода.

5 вариант) Пользователь вводит команды вида: «число<endline>». Далее это число передается от родительского процесса в дочерний. Дочерний процесс производит проверку на простоту. Если число составное, то в это число записывается в файл. Если число отрицательное или простое, то тогда дочерний и родительский процессы завершаются.

Общий метод и алгоритм решения

Использованные системные вызовы:

- shm_open(const char *name, int oflag, mode_t mode) — открывает или создает объект разделяемой памяти с указанным именем и флагами.
- ftruncate(int fd, off_t length) — изменяет размер объекта разделяемой памяти.
- mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset) — отображает объект разделяемой памяти в адресное пространство процесса.
- munmap(void *addr, size_t length) — удаляет отображение памяти.
- close(int fd) — закрывает файловый дескриптор объекта разделяемой памяти.
- shm_unlink(const char *name) — удаляет объект разделяемой памяти.

- fork() — создает новый процесс (дочерний процесс).
- execl(const char *path, const char *arg, ..., NULL) — заменяет текущий процесс новым, выполняя указанную программу.
- exit(int status) — завершает выполнение текущего процесса.
- wait(int *status) — ожидает завершения дочернего процесса.
- write(int fd, const void *buf, size_t count) — записывает данные в файловый дескриптор.
- read(int fd, void *buf, size_t count) — читает данные из файлового дескриптора.

Описание метода решения:

Программа реализует взаимодействие между процессами с использованием разделяемой памяти. Разделяемая память используется для передачи чисел между родительским и дочерним процессами, а также для передачи сигнала завершения работы программы.

Родительский процесс:

- Создает объект разделяемой памяти через shm_open.
- Устанавливает размер памяти с помощью ftruncate.
- Отображает объект разделяемой памяти в адресное пространство через mmap.
- Родительский процесс запрашивает у пользователя имя файла.
- Создает дочерний процесс с помощью fork и передает имя файла в качестве аргумента через execl.
- Родительский процесс записывает число, введенное пользователем, в разделяемую память.
- После записи числа устанавливает сигнал для дочернего процесса.
- Родительский процесс ожидает сигнал завершения от дочернего процесса через разделяемую память.
- Если сигнал завершения установлен, программа завершает выполнение.
- Удаляет объект разделяемой памяти через shm_unlink.

Дочерний процесс:

- После запуска через execl подключается к объекту разделяемой памяти, созданному родительским процессом.
- Читает числа из разделяемой памяти.
- Проверяет, является ли число простым. Если число простое или отрицательное:
- Отправляет сигнал завершения родительскому процессу.
- Завершает выполнение.
- Открывает указанный файл для записи чисел.
- Если число не является простым, записывает его в файл, разделяя пробелами.
- Удаляет отображение разделяемой памяти через munmap.

Отличия от подхода в лабораторной работе №1:

1. Отказ от каналов (pipe): взаимодействие между процессами теперь происходит через разделяемую память, что упрощает структуру кода и устраняет необходимость в дублировании файловых дескрипторов (dup2).

2. Эффективность передачи данных: разделяемая память позволяет обоим процессам напрямую работать с общим блоком памяти, исключая необходимость записи и чтения через файловые дескрипторы.

3. Сигналы завершения: используются флаги в разделяемой памяти для сигнализации завершения работы дочернего процесса.

4. Удаление ресурсов: родительский процесс после завершения работы удаляет объект разделяемой памяти через shm_unlink, освобождая ресурсы системы.

Код программы

parent.c

```c
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <sys/wait.h>
#include <semaphore.h>
#include <sys/stat.h>

#define SHM_NAME "/shared_memory"
#define SEM_REQUEST "/sem_request"
#define SEM_RESPONSE "/sem_response"

typedef struct {
    int number;
    int signal;
    char filename[128];
} SharedData;

void handle_error(const char *message) {
    write(STDERR_FILENO, message, strlen(message));
    exit(EXIT_FAILURE);
}

int main() {
    int shm_fd = shm_open(SHM_NAME, O_CREAT | O_RDWR, 0666);
    if (shm_fd == -1) {
        handle_error("Ошибка при создании разделяемой памяти\n");
    }

    if (ftruncate(shm_fd, sizeof(SharedData)) == -1) {
        handle_error("Ошибка при установке размера разделяемой памяти\n");
    }

    SharedData *data = mmap(NULL, sizeof(SharedData), PROT_READ | PROT_WRITE, MAP_SHARED,
shm_fd, 0);
    if (data == MAP_FAILED) {
        handle_error("Ошибка при отображении разделяемой памяти\n");
    }

    sem_t *sem_request = sem_open(SEM_REQUEST, O_CREAT, 0666, 0);
    sem_t *sem_response = sem_open(SEM_RESPONSE, O_CREAT, 0666, 0);

    if (sem_request == SEM_FAILED || sem_response == SEM_FAILED) {
        handle_error("Ошибка при создании семафоров\n");
    }
```

```
    write(STDOUT_FILENO, "Введите имя файла: ", 34);
    int n = read(STDIN_FILENO, data->filename, sizeof(data->filename));

    if (n <= 0) {
        handle_error("Ошибка при чтении имени файла\n");
    }
    data->filename[n - 1] = '\0';

    pid_t pid = fork();
    if (pid > 0) {
        char input[128];
        while (1) {
            write(STDOUT_FILENO, "Введите число: ", 28);
            n = read(STDIN_FILENO, input, sizeof(input));
            if (n <= 0) {
                handle_error("Ошибка при чтении числа\n");
            }

            input[n - 1] = '\0';
            char *endptr;
            data->number = strtol(input, &endptr, 10);
            if (endptr == input || *endptr != '\0') {
                handle_error("Ошибка: вводите только числа\n");
            }

            sem_post(sem_request);
            sem_wait(sem_response);

            if (data->signal == 1) {
                write(STDOUT_FILENO, "Программа завершена.\n", 40);
                break;
            }
        }

        wait(NULL);
        munmap(data, sizeof(SharedData));
        shm_unlink(SHM_NAME);
        sem_close(sem_request);
        sem_close(sem_response);
        sem_unlink(SEM_REQUEST);
        sem_unlink(SEM_RESPONSE);
    } else if (pid == 0) {
        execl("./child", "./child", NULL);
        handle_error("Ошибка при запуске дочернего процесса\n");
    } else {
        handle_error("Ошибка при вызове fork\n");
    }

    return 0;
}
```

child.c

```c
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <semaphore.h>
#include <sys/stat.h>

#define SHM_NAME "/shared_memory"
#define SEM_REQUEST "/sem_request"
#define SEM_RESPONSE "/sem_response"

typedef struct {
    int number;
    int signal;
    char filename[128];
} SharedData;

void handle_error(const char *message) {
    write(STDERR_FILENO, message, strlen(message));
    exit(EXIT_FAILURE);
}

int is_prime(int num) {
    if (num < 2) {
        return 0;
    }
    for (int i = 2; i * i <= num; i++) {
        if (num % i == 0) {
            return 0;
        }
    }
    return 1;
}

int int_to_str(int num, char *buf, int buf_size) {
    int len = 0;
    int temp = num;

    if (num < 0) {
        if (buf_size > 1) {
            buf[len++] = '-';
            num = -num;
        } else {
            return -1;
        }
    }

    do {
        temp /= 10;
        len++;
    } while (temp > 0);

    if (len >= buf_size) {
```

```
        return -1;
    }

    buf[len] = '\0';
    while (num > 0) {
        buf[--len] = (num % 10) + '0';
        num /= 10;
    }

    return strlen(buf);
}

int main() {
    int shm_fd = shm_open(SHM_NAME, O_RDWR, 0666);
    if (shm_fd == -1) {
        handle_error("Ошибка при открытии разделяемой памяти\n");
    }

    SharedData *data = mmap(NULL, sizeof(SharedData), PROT_READ | PROT_WRITE, MAP_SHARED,
shm_fd, 0);
    if (data == MAP_FAILED) {
        handle_error("Ошибка при отображении разделяемой памяти\n");
    }

    sem_t *sem_request = sem_open(SEM_REQUEST, 0);
    sem_t *sem_response = sem_open(SEM_RESPONSE, 0);

    if (sem_request == SEM_FAILED || sem_response == SEM_FAILED) {
        handle_error("Ошибка при открытии семафоров\n");
    }

    int file_fd = open(data->filename, O_WRONLY | O_CREAT | O_TRUNC, 0666);
    if (file_fd == -1) {
        handle_error("Ошибка при открытии файла\n");
    }

    while (1) {
        sem_wait(sem_request);

        int number = data->number;
        if (number < 0 || is_prime(number)) {
            data->signal = 1;
            sem_post(sem_response);
            break;
        } else {
            char buf[128];
            int len = int_to_str(number, buf, sizeof(buf));
            if (len == -1) {
                handle_error("Ошибка при преобразовании числа в строку\n");
            }

            if (write(file_fd, buf, len) == -1) {
                handle_error("Ошибка при записи в файл\n");
            }
```

```
            if (write(file_fd, " ", 1) == -1) {
                handle_error("Ошибка при записи пробела в файл\n");
            }

            data->signal = 0;
            sem_post(sem_response);
        }
    }

    close(file_fd);
    munmap(data, sizeof(SharedData));
    return 0;
}
```

Протокол работы программы

Тестирование:

migermanenko@MacBook-Pro-Matvej src % ./parent

Введите имя файла: 123.txt

Введите число: 44

Введите число: 13

Программа завершена.

migermanenko@MacBook-Pro-Matvej src % ./parent

Введите имя файла: test.txt

Введите число: 66

Введите число: 252

Введите число: 14

Введите число: 35

Введите число: 1

Введите число: 2

Программа завершена.

migermanenko@MacBook-Pro-Matvej src % ./parent

Введите имя файла: 3485.txt

Введите число: 14

Введите число: 2f

Ошибка: вводите только числа.

Dtruss:

migermanenko@MacBook-Pro-Matvej src % sudo dtruss -f ./parent

PID/THRD SYSCALL(args) = return

Введите имя файла: 1124/0x2ecd: fork() = 0 0

1124/0x2ecd: munmap(0x102468000, 0x84000) = 0 0

1124/0x2ecd: munmap(0x1024EC000, 0x8000) = 0 0

1124/0x2ecd: munmap(0x1024F4000, 0x4000) = 0 0

1124/0x2ecd: munmap(0x1024F8000, 0x4000) = 0 0

1124/0x2ecd: munmap(0x1024FC000, 0x48000) = 0 0

1124/0x2ecd: munmap(0x102544000, 0x4C000) = 0 0

1124/0x2ecd: crossarch_trap(0x0, 0x0, 0x0) = -1 Err#45

1124/0x2ecd: open(".\0", 0x100000, 0x0) = 3 0

1124/0x2ecd: fcntl(0x3, 0x32, 0x16DB330C8) = 0 0

1124/0x2ecd: close(0x3) = 0 0

1124/0x2ecd: fsgetpath(0x16DB330D8, 0x400, 0x16DB330B8) = 51 0

1124/0x2ecd: fsgetpath(0x16DB330E8, 0x400, 0x16DB330C8) = 14 0

1124/0x2ecd: csrctl(0x0, 0x16DB334EC, 0x4) = -1 Err#1

1124/0x2ecd: __mac_syscall(0x18D12FC12, 0x2, 0x16DB33430) = 0 0

1124/0x2ecd: csrctl(0x0, 0x16DB334DC, 0x4) = -1 Err#1

1124/0x2ecd: __mac_syscall(0x18D12CA45, 0x5A, 0x16DB33470) = 0 0

1124/0x2ecd: sysctl([unknown, 3, 0, 0, 0, 0] (2), 0x16DB329D8, 0x16DB329D0, 0x18D12E738, 0xD) = 0 0

1124/0x2ecd: sysctl([CTL_KERN, 157, 0, 0, 0, 0] (2), 0x16DB32A88, 0x16DB32A80, 0x0, 0x0) = 0 0

1124/0x2ecd: open("/\0", 0x20100000, 0x0) = 3 0

1124/0x2ecd: openat(0x3, "System/Cryptexes/OS\0", 0x100000, 0x0) = 4 0

1124/0x2ecd: dup(0x4, 0x0, 0x0) = 5 0

1124/0x2ecd: fstatat64(0x4, 0x16DB32561, 0x16DB324D0) = 0 0

1124/0x2ecd: openat(0x4, "System/Library/dyld/\0", 0x100000, 0x0) = 6 0

1124/0x2ecd: fcntl(0x6, 0x32, 0x16DB32560) = 0 0

1124/0x2ecd: dup(0x6, 0x0, 0x0) = 7 0

1124/0x2ecd: dup(0x5, 0x0, 0x0) = 8 0

1124/0x2ecd: close(0x3) = 0 0

1124/0x2ecd: close(0x5) = 0 0

1124/0x2ecd: close(0x4) = 0 0

1124/0x2ecd: close(0x6) = 0 0

1124/0x2ecd: __mac_syscall(0x18D12FC12, 0x2, 0x16DB32F50) = 0 0

1124/0x2ecd: shared_region_check_np(0x16DB32B70, 0x0, 0x0) = 0 0

1124/0x2ecd: fsgetpath(0x16DB330F0, 0x400, 0x16DB33018) = 82 0

1124/0x2ecd: fcntl(0x8, 0x32, 0x16DB330F0) = 0 0

1124/0x2ecd: close(0x8) = 0 0

1124/0x2ecd: close(0x7) = 0 0

1124/0x2ecd: getfsstat64(0x0, 0x0, 0x2) = 12 0

1124/0x2ecd: getfsstat64(0x1022C8050, 0x65A0, 0x2) = 12 0

1124/0x2ecd: getattrlist("/\0", 0x16DB33030, 0x16DB32FA0) = 0 0

1124/0x2ecd:
stat64("/System/Volumes/Preboot/Cryptexes/OS/System/Library/dyld/dyld_shared_cache_arm64e\0", 0x16DB33390, 0x0) = 0 0

dtrace: error on enabled probe ID 1696 (ID 845: syscall::stat64:return): invalid address (0x0) in action #12 at DIF offset 12

1124/0x2ecd: stat64("/Users/migermanenko/Documents/C/OC/Lab3/src/parent\0", 0x16DB32840, 0x0) = 0 0

1124/0x2ecd: open("/Users/migermanenko/Documents/C/OC/Lab3/src/parent\0", 0x0, 0x0) = 3 0

1124/0x2ecd: mmap(0x0, 0x8648, 0x1, 0x40002, 0x3, 0x0) = 0x1022C8000 0

1124/0x2ecd: fcntl(0x3, 0x32, 0x16DB32958) = 0 0

1124/0x2ecd: close(0x3) = 0 0

1124/0x2ecd: munmap(0x1022C8000, 0x8648) = 0 0

1124/0x2ecd: stat64("/Users/migermanenko/Documents/C/OC/Lab3/src/parent\0", 0x16DB32DB0, 0x0) = 0 0

1124/0x2ecd: stat64("/usr/lib/libSystem.B.dylib\0", 0x16DB31D00, 0x0) = -1 Err#2

1124/0x2ecd: stat64("/System/Volumes/Preboot/Cryptexes/OS/usr/lib/libSystem.B.dylib\0", 0x16DB31CB0, 0x0) = -1 Err#2

1124/0x2ecd: open("/Users/migermanenko/Documents/C/OC/Lab3/src/parent\0", 0x0, 0x0) = 3 0

1124/0x2ecd: __mac_syscall(0x18D12FC12, 0x2, 0x16DB30400) = 0 0

1124/0x2ecd: map_with_linking_np(0x16DB30280, 0x1, 0x16DB302B0) = 0 0

1124/0x2ecd: close(0x3) = 0 0

1124/0x2ecd: mprotect(0x1022C0000, 0x4000, 0x1) = 0 0

```
1124/0x2ecd:  open("/dev/dtracehelper\0", 0x2, 0x0)           = 3 0
1124/0x2ecd:  ioctl(0x3, 0x80086804, 0x16DB2F9B8)             = 0 0
1124/0x2ecd:  close(0x3)              = 0 0
1124/0x2ecd:  shared_region_check_np(0xFFFFFFFFFFFFFFFF, 0x0, 0x0)            = 0 0
1124/0x2ecd:  access("/AppleInternal/XBS/.isChrooted\0", 0x0, 0x0)            = -1 Err#2
1124/0x2ecd:  bsdthread_register(0x18D4320F4, 0x18D4320E8, 0x4000)            = 1073746399 0
1124/0x2ecd:  getpid(0x0, 0x0, 0x0)           = 1124 0
1124/0x2ecd:  shm_open(0x18D2C9F41, 0x0, 0xFFFFFFFF8D470000)          = 3 0
1124/0x2ecd:  fstat64(0x3, 0x16DB30030, 0x0)          = 0 0
1124/0x2ecd:  mmap(0x0, 0x8000, 0x1, 0x40001, 0x3, 0x0)               = 0x1022D0000 0
1124/0x2ecd:  close(0x3)              = 0 0
1124/0x2ecd:  csops(0x464, 0x0, 0x16DB3016C)          = 0 0
1124/0x2ecd:  ioctl(0x2, 0x4004667A, 0x16DB300DC)             = 0 0
1124/0x2ecd:  mprotect(0x1022E0000, 0x4000, 0x0)              = 0 0
1124/0x2ecd:  mprotect(0x1022EC000, 0x4000, 0x0)              = 0 0
1124/0x2ecd:  mprotect(0x1022F0000, 0x4000, 0x0)              = 0 0
1124/0x2ecd:  mprotect(0x1022FC000, 0x4000, 0x0)              = 0 0
1124/0x2ecd:  mprotect(0x102300000, 0x4000, 0x0)              = 0 0
1124/0x2ecd:  mprotect(0x10230C000, 0x4000, 0x0)              = 0 0
1124/0x2ecd:  mprotect(0x1022D8000, 0xC8, 0x1)        = 0 0
1124/0x2ecd:  mprotect(0x1022D8000, 0xC8, 0x3)        = 0 0
1124/0x2ecd:  mprotect(0x1022D8000, 0xC8, 0x1)        = 0 0
1124/0x2ecd:  mprotect(0x102310000, 0x4000, 0x1)              = 0 0
1124/0x2ecd:  mprotect(0x102314000, 0xC8, 0x1)        = 0 0
1124/0x2ecd:  mprotect(0x102314000, 0xC8, 0x3)        = 0 0
1124/0x2ecd:  mprotect(0x102314000, 0xC8, 0x1)        = 0 0
1124/0x2ecd:  mprotect(0x1022D8000, 0xC8, 0x3)        = 0 0
1124/0x2ecd:  mprotect(0x1022D8000, 0xC8, 0x1)        = 0 0
1124/0x2ecd:  mprotect(0x102310000, 0x4000, 0x3)              = 0 0
1124/0x2ecd:  mprotect(0x102310000, 0x4000, 0x1)              = 0 0
1124/0x2ecd:  issetugid(0x0, 0x0, 0x0)        = 0 0
1124/0x2ecd:  getentropy(0x16DB2F738, 0x20, 0x0)              = 0 0
```

1124/0x2ecd: getattrlist("/Users/migermanenko/Documents/C/OC/Lab3/src/parent\0", 0x16DB2FFD0, 0x16DB2FFEC) = 0 0

1124/0x2ecd: access("/Users/migermanenko/Documents/C/OC/Lab3/src\0", 0x4, 0x0) = 0 0

1124/0x2ecd: open("/Users/migermanenko/Documents/C/OC/Lab3/src\0", 0x0, 0x0) = 3 0

1124/0x2ecd: fstat64(0x3, 0x148604500, 0x0) = 0 0

1124/0x2ecd: csrctl(0x0, 0x16DB301BC, 0x4) = -1 Err#1

1124/0x2ecd: fgetattrlist(0x3, 0x16DB30260, 0x16DB301E0) = 0 0

1124/0x2ecd: __mac_syscall(0x199397505, 0x2, 0x16DB301E0) = 0 0

1124/0x2ecd: fcntl(0x3, 0x32, 0x16DB2FEB8) = 0 0

1124/0x2ecd: close(0x3) = 0 0

1124/0x2ecd: open("/Users/migermanenko/Documents/C/OC/Lab3/src/Info.plist\0", 0x0, 0x0) = -1 Err#2

1124/0x2ecd: proc_info(0x2, 0x464, 0xD) = 64 0

1124/0x2ecd: csops_audittoken(0x464, 0x10, 0x16DB30240) = 0 0

1124/0x2ecd: sysctl([unknown, 3, 0, 0, 0, 0] (2), 0x16DB30598, 0x16DB30590, 0x190B5CD3A, 0x15) = 0 0

1124/0x2ecd: sysctl([CTL_KERN, 155, 0, 0, 0, 0] (2), 0x16DB30628, 0x16DB30620, 0x0, 0x0) = 0 0

1124/0x2ecd: shm_open(0x1022BFCD4, 0x202, 0x1B6) = 3 0

1124/0x2ecd: ftruncate(0x3, 0x88, 0x0) = 0 0

1124/0x2ecd: mmap(0x0, 0x88, 0x3, 0x40001, 0x3, 0x0) = 0x10231C000 0

1124/0x2ecd: sem_open(0x1022BFDD8, 0x200, 0x1B6) = 4 0

1124/0x2ecd: sem_open(0x1022BFDE5, 0x200, 0x1B6) = 5 0

1124/0x2ecd: write(0x1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \320\270\320\274\321\217 \321\204\320\260\320\271\320\273\320\260: \0", 0x22) = 34 0

test.txt

Введите число: 1124/0x2ecd: read(0x0, "test.txt\n\0", 0x80) = 9 0

1124/0x2ecd: fork() = 1128 0

1128/0x2f7e: fork() = 0 0

1128/0x2f7e: thread_selfid(0x0, 0x0, 0x0) = 12158 0

1128/0x2f7e: bsdthread_register(0x18D4320F4, 0x18D4320E8, 0x4000) = -1 Err#22

1124/0x2ecd: write(0x1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \321\207\320\270\321\201\320\273\320\276: \0", 0x1C) = 28 0

1128/0x2f7e: mprotect(0x102314000, 0xC8, 0x3) = 0 0

1128/0x2f7e: mprotect(0x102314000, 0xC8, 0x1) = 0 0

dtrace: error on enabled probe ID 1694 (ID 287: syscall::execve:return): invalid address (0x1022bff2e) in action #12 at DIF offset 12

1128/0x2f7f: fork() = 0 0

1128/0x2f7f: mprotect(0x102710000, 0x8000, 0x1) = 0 0

1128/0x2f7f: thread_selfid(0x0, 0x0, 0x0) = 12159 0

1128/0x2f7f: crossarch_trap(0x0, 0x0, 0x0) = -1 Err#45

1128/0x2f7f: shared_region_check_np(0x16DA47790, 0x0, 0x0) = 0 0

1128/0x2f7f: thread_selfid(0x0, 0x0, 0x0) = 12159 0

1128/0x2f7f: getpid(0x0, 0x0, 0x0) = 1128 0

1128/0x2f7f: proc_info(0xF, 0x468, 0x0) = 0 0

1128/0x2f7f: munmap(0x10268C000, 0x84000) = 0 0

1128/0x2f7f: munmap(0x102710000, 0x8000) = 0 0

1128/0x2f7f: munmap(0x102718000, 0x4000) = 0 0

1128/0x2f7f: munmap(0x10271C000, 0x4000) = 0 0

1128/0x2f7f: munmap(0x102720000, 0x48000) = 0 0

1128/0x2f7f: munmap(0x102768000, 0x4C000) = 0 0

1128/0x2f7f: crossarch_trap(0x0, 0x0, 0x0) = -1 Err#45

1128/0x2f7f: open(".\0", 0x100000, 0x0) = 3 0

1128/0x2f7f: fcntl(0x3, 0x32, 0x16DA370C8) = 0 0

1128/0x2f7f: close(0x3) = 0 0

1128/0x2f7f: fsgetpath(0x16DA370D8, 0x400, 0x16DA370B8) = 50 0

1128/0x2f7f: fsgetpath(0x16DA370E8, 0x400, 0x16DA370C8) = 14 0

1128/0x2f7f: csrctl(0x0, 0x16DA374EC, 0x4) = -1 Err#1

1128/0x2f7f: __mac_syscall(0x18D12FC12, 0x2, 0x16DA37430) = 0 0

1128/0x2f7f: csrctl(0x0, 0x16DA374DC, 0x4) = -1 Err#1

1128/0x2f7f: __mac_syscall(0x18D12CA45, 0x5A, 0x16DA37470) = 0 0

1128/0x2f7f: sysctl([unknown, 3, 0, 0, 0, 0] (2), 0x16DA369D8, 0x16DA369D0, 0x18D12E738, 0xD) = 0 0

1128/0x2f7f: sysctl([CTL_KERN, 157, 0, 0, 0, 0] (2), 0x16DA36A88, 0x16DA36A80, 0x0, 0x0) = 0 0

1128/0x2f7f: open("/\0", 0x20100000, 0x0) = 3 0

1128/0x2f7f: openat(0x3, "System/Cryptexes/OS\0", 0x100000, 0x0) = 6 0

1128/0x2f7f: dup(0x6, 0x0, 0x0) = 7 0

```
1128/0x2f7f:  fstatat64(0x6, 0x16DA36561, 0x16DA364D0)        = 0 0

1128/0x2f7f:  openat(0x6, "System/Library/dyld/\0", 0x100000, 0x0)            = 8 0

1128/0x2f7f:  fcntl(0x8, 0x32, 0x16DA36560)           = 0 0

1128/0x2f7f:  dup(0x8, 0x0, 0x0)              = 9 0

1128/0x2f7f:  dup(0x7, 0x0, 0x0)              = 10 0

1128/0x2f7f:  close(0x3)              = 0 0

1128/0x2f7f:  close(0x7)              = 0 0

1128/0x2f7f:  close(0x6)              = 0 0

1128/0x2f7f:  close(0x8)              = 0 0

1128/0x2f7f:  __mac_syscall(0x18D12FC12, 0x2, 0x16DA36F50)            = 0 0

1128/0x2f7f:  shared_region_check_np(0x16DA36B70, 0x0, 0x0)           = 0 0

1128/0x2f7f:  fsgetpath(0x16DA370F0, 0x400, 0x16DA37018)              = 82 0

1128/0x2f7f:  fcntl(0xA, 0x32, 0x16DA370F0)           = 0 0

1128/0x2f7f:  close(0xA)              = 0 0

1128/0x2f7f:  close(0x9)              = 0 0

1128/0x2f7f:  getfsstat64(0x0, 0x0, 0x2)              = 12 0

1128/0x2f7f:  getfsstat64(0x1023C4050, 0x65A0, 0x2)           = 12 0

1128/0x2f7f:  getattrlist("/\0", 0x16DA37030, 0x16DA36FA0)            = 0 0

1128/0x2f7f:
  stat64("/System/Volumes/Preboot/Cryptexes/OS/System/Library/dyld/dyld_shared_cache_arm64
  e\0", 0x16DA37390, 0x0)          = 0 0

dtrace: error on enabled probe ID 1696 (ID 845: syscall::stat64:return): invalid address (0x0) in
  action #12 at DIF offset 12

1128/0x2f7f:  stat64("/Users/migermanenko/Documents/C/OC/Lab3/src/child\0", 0x16DA36840,
  0x0)          = 0 0

1128/0x2f7f:  open("/Users/migermanenko/Documents/C/OC/Lab3/src/child\0", 0x0, 0x0)
  = 3 0

1128/0x2f7f:  mmap(0x0, 0x8558, 0x1, 0x40002, 0x3, 0x0)               = 0x1023C4000 0

1128/0x2f7f:  fcntl(0x3, 0x32, 0x16DA36958)           = 0 0

1128/0x2f7f:  close(0x3)              = 0 0

1128/0x2f7f:  munmap(0x1023C4000, 0x8558)             = 0 0

1128/0x2f7f:  stat64("/Users/migermanenko/Documents/C/OC/Lab3/src/child\0", 0x16DA36DB0,
  0x0)          = 0 0

1128/0x2f7f:  stat64("/usr/lib/libSystem.B.dylib\0", 0x16DA35D00, 0x0)         = -1 Err#2
```

1128/0x2f7f: stat64("/System/Volumes/Preboot/Cryptexes/OS/usr/lib/libSystem.B.dylib\0", 0x16DA35CB0, 0x0) = -1 Err#2

1128/0x2f7f: open("/Users/migermanenko/Documents/C/OC/Lab3/src/child\0", 0x0, 0x0) = 3 0

1128/0x2f7f: __mac_syscall(0x18D12FC12, 0x2, 0x16DA34400) = 0 0

1128/0x2f7f: map_with_linking_np(0x16DA342C0, 0x1, 0x16DA342F0) = 0 0

1128/0x2f7f: close(0x3) = 0 0

1128/0x2f7f: mprotect(0x1023BC000, 0x4000, 0x1) = 0 0

1128/0x2f7f: open("/dev/dtracehelper\0", 0x2, 0x0) = 3 0

1128/0x2f7f: ioctl(0x3, 0x80086804, 0x16DA339B8) = 0 0

1128/0x2f7f: close(0x3) = 0 0

1128/0x2f7f: shared_region_check_np(0xFFFFFFFFFFFFFFFF, 0x0, 0x0) = 0 0

1128/0x2f7f: access("/AppleInternal/XBS/.isChrooted\0", 0x0, 0x0) = -1 Err#2

1128/0x2f7f: bsdthread_register(0x18D4320F4, 0x18D4320E8, 0x4000) = 1073746399 0

1128/0x2f7f: getpid(0x0, 0x0, 0x0) = 1128 0

1128/0x2f7f: shm_open(0x18D2C9F41, 0x0, 0xFFFFFFFF9AE58000) = 3 0

1128/0x2f7f: fstat64(0x3, 0x16DA34030, 0x0) = 0 0

1128/0x2f7f: mmap(0x0, 0x8000, 0x1, 0x40001, 0x3, 0x0) = 0x1023CC000 0

1128/0x2f7f: close(0x3) = 0 0

1128/0x2f7f: csops(0x468, 0x0, 0x16DA3416C) = 0 0

1128/0x2f7f: ioctl(0x2, 0x4004667A, 0x16DA340DC) = 0 0

1128/0x2f7f: mprotect(0x1023DC000, 0x4000, 0x0) = 0 0

1128/0x2f7f: mprotect(0x1023E8000, 0x4000, 0x0) = 0 0

1128/0x2f7f: mprotect(0x1023EC000, 0x4000, 0x0) = 0 0

1128/0x2f7f: mprotect(0x1023F8000, 0x4000, 0x0) = 0 0

1128/0x2f7f: mprotect(0x1023FC000, 0x4000, 0x0) = 0 0

1128/0x2f7f: mprotect(0x102408000, 0x4000, 0x0) = 0 0

1128/0x2f7f: mprotect(0x1023D4000, 0xC8, 0x1) = 0 0

1128/0x2f7f: mprotect(0x1023D4000, 0xC8, 0x3) = 0 0

1128/0x2f7f: mprotect(0x1023D4000, 0xC8, 0x1) = 0 0

1128/0x2f7f: mprotect(0x10240C000, 0x4000, 0x1) = 0 0

1128/0x2f7f: mprotect(0x102410000, 0xC8, 0x1) = 0 0

1128/0x2f7f: mprotect(0x102410000, 0xC8, 0x3) = 0 0

1128/0x2f7f: mprotect(0x102410000, 0xC8, 0x1) = 0 0

1128/0x2f7f: mprotect(0x1023D4000, 0xC8, 0x3) = 0 0

1128/0x2f7f: mprotect(0x1023D4000, 0xC8, 0x1) = 0 0

1128/0x2f7f: mprotect(0x10240C000, 0x4000, 0x3) = 0 0

1128/0x2f7f: mprotect(0x10240C000, 0x4000, 0x1) = 0 0

1128/0x2f7f: issetugid(0x0, 0x0, 0x0) = 0 0

1128/0x2f7f: getentropy(0x16DA33738, 0x20, 0x0) = 0 0

1128/0x2f7f: getattrlist("/Users/migermanenko/Documents/C/OC/Lab3/src/child\0", 0x16DA33FD0, 0x16DA33FEC) = 0 0

1128/0x2f7f: access("/Users/migermanenko/Documents/C/OC/Lab3/src\0", 0x4, 0x0) = 0 0

1128/0x2f7f: open("/Users/migermanenko/Documents/C/OC/Lab3/src\0", 0x0, 0x0) = 3 0

1128/0x2f7f: fstat64(0x3, 0x14B604500, 0x0) = 0 0

1128/0x2f7f: csrctl(0x0, 0x16DA341BC, 0x4) = -1 Err#1

1128/0x2f7f: fgetattrlist(0x3, 0x16DA34260, 0x16DA341E0) = 0 0

1128/0x2f7f: __mac_syscall(0x199397505, 0x2, 0x16DA341E0) = 0 0

1128/0x2f7f: fcntl(0x3, 0x32, 0x16DA33EB8) = 0 0

1128/0x2f7f: close(0x3) = 0 0

1128/0x2f7f: open("/Users/migermanenko/Documents/C/OC/Lab3/src/Info.plist\0", 0x0, 0x0) = -1 Err#2

1128/0x2f7f: proc_info(0x2, 0x468, 0xD) = 64 0

1128/0x2f7f: csops_audittoken(0x468, 0x10, 0x16DA34240) = 0 0

1128/0x2f7f: sysctl([unknown, 3, 0, 0, 0, 0] (2), 0x16DA34598, 0x16DA34590, 0x190B5CD3A, 0x15) = 0 0

1128/0x2f7f: sysctl([CTL_KERN, 155, 0, 0, 0, 0] (2), 0x16DA34628, 0x16DA34620, 0x0, 0x0) = 0 0

1128/0x2f7f: shm_open(0x1023BBDB8, 0x2, 0x1B6) = 3 0

1128/0x2f7f: mmap(0x0, 0x88, 0x3, 0x40001, 0x3, 0x0) = 0x102418000 0

1128/0x2f7f: sem_open(0x1023BBE61, 0x0, 0x1B6) = 6 0

1128/0x2f7f: sem_open(0x1023BBE6E, 0x0, 0x1B6) = 7 0

1128/0x2f7f: open("test.txt\0", 0x601, 0x1B6) = 8 0

10

Введите число: 1124/0x2ecd: read(0x0, "10\n\0", 0x80) = 3 0

1124/0x2ecd: sem_post(0x4, 0x0, 0x0) = 0 0

1128/0x2f7f: sem_wait(0x6, 0x0, 0x0) = 0 0

1128/0x2f7f: write(0x8, "10\0", 0x2) = 2 0

1128/0x2f7f: write(0x8, " \0", 0x1) = 1 0

1128/0x2f7f: sem_post(0x7, 0x0, 0x0) = 0 0

1124/0x2ecd: sem_wait(0x5, 0x0, 0x0) = 0 0

1124/0x2ecd: write(0x1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265
\321\207\320\270\321\201\320\273\320\276: \0", 0x1C) = 28 0

55

Введите число: 1124/0x2ecd: read(0x0, "55\n\0", 0x80) = 3 0

1124/0x2ecd: sem_post(0x4, 0x0, 0x0) = 0 0

1128/0x2f7f: sem_wait(0x6, 0x0, 0x0) = 0 0

1128/0x2f7f: write(0x8, "55\0", 0x2) = 2 0

1128/0x2f7f: write(0x8, " \0", 0x1) = 1 0

1128/0x2f7f: sem_post(0x7, 0x0, 0x0) = 0 0

1124/0x2ecd: sem_wait(0x5, 0x0, 0x0) = 0 0

1124/0x2ecd: write(0x1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265
\321\207\320\270\321\201\320\273\320\276: \0", 0x1C) = 28 0

66

Введите число: 1124/0x2ecd: read(0x0, "66\n\0", 0x80) = 3 0

1124/0x2ecd: sem_post(0x4, 0x0, 0x0) = 0 0

1128/0x2f7f: sem_wait(0x6, 0x0, 0x0) = 0 0

1128/0x2f7f: write(0x8, "66\0", 0x2) = 2 0

1128/0x2f7f: write(0x8, " \0", 0x1) = 1 0

1128/0x2f7f: sem_post(0x7, 0x0, 0x0) = 0 0

1124/0x2ecd: sem_wait(0x5, 0x0, 0x0) = 0 0

1124/0x2ecd: write(0x1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265
\321\207\320\270\321\201\320\273\320\276: \0", 0x1C) = 28 0

2

Программа завершена.

1124/0x2ecd: read(0x0, "2\n\0", 0x80) = 2 0

1124/0x2ecd: sem_post(0x4, 0x0, 0x0) = 0 0

1128/0x2f7f: sem_wait(0x6, 0x0, 0x0) = 0 0

1128/0x2f7f: sem_post(0x7, 0x0, 0x0) = 0 0

1124/0x2ecd: sem_wait(0x5, 0x0, 0x0) = 0 0

1124/0x2ecd: write(0x1,
"\320\237\321\200\320\276\320\263\321\200\320\260\320\274\320\274\320\260
\320\267\320\260\320\262\320\265\321\200\321\210\320\265\320\275\320\260.\n\0", 0x28)
= 40 0

1128/0x2f7f: close(0x8) = 0 0

1128/0x2f7f: munmap(0x102418000, 0x88) = 0 0

1124/0x2ecd: wait4(0xFFFFFFFFFFFFFFFF, 0x0, 0x0) = 1128 0

1124/0x2ecd: munmap(0x10231C000, 0x88) = 0 0

1124/0x2ecd: shm_unlink(0x1022BFCD4, 0x0, 0x0) = 0 0

1124/0x2ecd: sem_close(0x4, 0x0, 0x0) = 0 0

1124/0x2ecd: sem_close(0x5, 0x0, 0x0) = 0 0

1124/0x2ecd: sem_unlink(0x1022BFDD8, 0x0, 0x0) = 0 0

1124/0x2ecd: sem_unlink(0x1022BFDE5, 0x0, 0x0) = 0 0

Вывод

Использование разделяемой памяти вместо каналов (pipe) значительно упрощает взаимодействие между процессами, устраняет необходимость дублирования файловых дескрипторов и системных вызовов для записи/чтения. Этот метод повышает эффективность передачи данных, так как оба процесса работают с общим блоком памяти напрямую. Кроме того, управляемость ресурсов улучшается за счет явного удаления объектов разделяемой памяти после завершения работы.