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-210Б-23

Студент: Воронухин Н.А.

Преподаватель: Бахарев В.Д.

Оценка: ____________________

Дата: 10.12.24

Москва, 2024

Постановка задачи

Вариант 1.

Реализовать межпроцессорное взаимодействие с использованием shared memory и memory mapping и семафоров.

Пользователь вводит команды вида: «число число число<endline>». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс производит деление первого числа, на последующие, а результат выводит в файл. Если происходит деление на 0, то тогда дочерний и родительский процесс завершают свою работу. Проверка деления на 0 должна осуществляться на стороне дочернего процесса. Числа имеют тип int. Количество чисел может быть произвольным.

Общий метод и алгоритм решения

Использованные системные вызовы:

- void exit(int status); - приводит к обычному завершению программы, и величина status возвращается процессу-родителю.
- exit_group – системный вызов, завершающий все потоки процесса.
- clone – используется для создания дочернего процесса или потока в Linux
- futex – используется для создания фьютекса – легковесной версии мьютекса.
- 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. Последний параметр (адрес) необязателен, и обычно бывает равен 0. Настоящее местоположение отраженных данных возвращается самой функцией mmap, и никогда не бывает равным 0.
- int munmap(void **start*, size_t *length*); - удаляет отображение для указанного адресного диапазона
- pid_t fork(void); – создает дочерний процесс.
- ssize_t write(int fd, const void buf[.count], size_t count); - пишет count байт из буфера buf в файл, на который ссылается файловый дескриптор fd.
- ssize_t read(int fd, void buf[.count], size_t count); - считывает count байт в буфер buf из файла, на который ссылается файловый дескриптор fd
- pid_t wait(int *status); - приостанавливает выполнение текущего процесса до тех пор, пока дочерний процесс не завершится, или до появления сигнала, который либо завершает текущий процесс, либо требует вызвать функцию-обработчик.
- int shm_open(const char *name, int oflag, mode_t mode); - создает и открывает новый (или открывает уже существующий) объект разделяемой памяти POSIX. Объект разделяемой памяти POSIX - это обработчик, используемый несвязанными процессами для исполнения mmap на одну область разделяемой памяти.
- int ftruncate(int fd, off_t length); -устанавливает длину файла с файловым дескриптором fd в length байт
- int shm_unlink(const char *name); - удаляет имя объекта разделяемой памяти
- sem_t *sem_open(const char *name, int oflag); - создаёт новый семафор или открывает уже существующий

- int sem_wait(sem_t *sem); - уменьшает (блокирует) семафор, на который указывает *sem*.
- int sem_post(sem_t *sem); - величивает (разблокирует) семафор, на который указывает *sem*.
- int sem_close(sem_t *sem); - закрывает именованный семафор, на который указывает sem, позволяя освободить все ресурсы, которые система выделила под семафор вызывающему процессу.
- int sem_unlink(const char *name); - удаляет именованный семафор, на который ссылается name. Имя семафора удаляется немедленно. Семафор уничтожается после того, как все остальные процессы, в которых он открыт, закроют его.

Родительский процесс создает объект shared memory, открывает семафоры, читает данные с потока ввода, и передает их дочернему процессу, вычисляющему значения. Затем дочерний процесс возвращает результат так же с помощью разделяемой памяти. Всё общение через разделяемую память регулируется семафорами.

Код программы

lab3_defines.h

```
#ifndef _LAB3_DEFINES

    #define _LAB3_DEFINES


    #include <semaphore.h>


    static const int STR_BUF_SIZE = 500;

    static const int SHM_SIZE = STR_BUF_SIZE * sizeof(char);

    const char SHM_NAME[] = "/shared_memory";

    const char SEM_WRITE[] = "/sem_write_";

    const char SEM_READ[] = "/sem_read_";


    enum ERR_FLG

    {

        READ_ERR = 1,
        WRITE_ERR,
        WRONG_INSTRUCTION_ERR,
        SHM_ERROR,
        SEM_ERROR,


        WRONG_FILE_NAME_ERR,
        OPEN_FILE_ERR,
        CREATE_PROCESS_ERR,
        READ_PATH_ERR,
        CONVERT_ERR,
        CAT_ERR,
        EXEC_ERR,
        CHILD_ERROR = 1000,
    };


    enum MEM_SIGNALS
    {
        STOP_MES,
        READY_TO_READ,
        ERROR_MES_FROM_CHILD,
        ERROR_MES_FROM_PARENT,
        READY_TO_WRITE,
    };


    struct shm_handler
    {
        int shm_fd;
        char *shm_ptr;
        sem_t *sem_wr;
        sem_t *sem_rd;
    };


#endif
```

parent.c

```
#include <unistd.h>

#include <sys/types.h>

#include <fcntl.h>

#include <sys/stat.h>

#include <stdlib.h>

#include <ctype.h>

#include <sys/wait.h>

#include <stdio.h>

#include <semaphore.h>

#include <string.h>

#include <sys/mman.h>

#include "lab3_defines.h"


static const int FILE_NAME_BUF_SIZE = 15;

static char CHILDE_PROGRAM_NAME[] = "childe"; // child

static const int PATH_SIZE = 1024;


void exitParent(int, const char* const, int);


int main() {


    char progpath[PATH_SIZE];

    ssize_t len = readlink("/proc/self/exe", progpath, sizeof(progpath) / sizeof(char)
- sizeof(char));

    if (len == -1) {

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

        exitParent(READ_PATH_ERR, mes, sizeof(mes));

    }

    while (progpath[len] != '/')

        --len;
```

```
progpath[len] = '\0';


char buff[FILE_NAME_BUF_SIZE];
int nameFileLen = 0;
if ((nameFileLen = read(STDIN_FILENO, buff, sizeof(buff))) == -1) {
    const char mes[] = "Error: failed to read from STDIN\n";
    exitParent(READ_ERR, mes, sizeof(mes));
}
else if (buff[nameFileLen - 1] != '\n') {
    const char mes[] = "Error: file name too long\n";
    exitParent(WRONG_FILE_NAME_ERR, mes, sizeof(mes));
}
buff[nameFileLen - 1] = '\0';


int fd = open(buff, O_WRONLY | O_CREAT | O_TRUNC, S_IRWXU | S_IRWXO);
if (fd == -1) {
    const char mes[] = "Error: failed to open file\n";
    exitParent(OPEN_FILE_ERR, mes, sizeof(mes));
}
/*
    int pToChilde[2];
    if (pipe(pToChilde)) {
        close(fd);
        const char mes[] = "Error: failed to create pipe\n";
        exitParent(CREATE_PIPE_ERR, mes, sizeof(mes));
    }
    int pToParent[2];
    if (pipe(pToParent)) {
        close(fd);
        close(pToChilde[1]);
        close(pToChilde[0]);
```

```
                const char mes[] = "Error: failed to create pipe\n";

                exitParent(CREATE_PIPE_ERR, mes, sizeof(mes));

            }

        */


        struct shm_handler shm_obj;

        shm_obj.shm_fd = shm_open(SHM_NAME, O_CREAT | O_RDWR, 0666);

        if (shm_obj.shm_fd == -1){

            close(fd);

            const char mes[] = "Error shm_open.\n";

            exitParent(SHM_ERROR, mes, sizeof(mes));

        }


        if (ftruncate(shm_obj.shm_fd, BUFSIZ) == -1){

            shm_unlink(SHM_NAME);

            close(fd);

            const char mes[] = "Error ftruncate.\n";

            exitParent(SHM_ERROR, mes, sizeof(mes));

        }


        shm_obj.shm_ptr = (char*)mmap(0, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED,
shm_obj.shm_fd, 0);

        if (shm_obj.shm_ptr == MAP_FAILED) {

            shm_unlink(SHM_NAME);

            close(fd);

            const char mes[] = "Error mmap.\n";

            exitParent(SHM_ERROR, mes, sizeof(mes));

        }


        shm_obj.sem_wr = sem_open(SEM_WRITE, O_CREAT, 0666, 0);

        if (shm_obj.sem_wr == SEM_FAILED)

        {

            munmap(shm_obj.shm_ptr, SHM_SIZE);
            shm_unlink(SHM_NAME);

            close(fd);

            const char mes[] = "Error sem_open.\n";

            exitParent(SEM_ERROR, mes, sizeof(mes));

        }

        shm_obj.sem_rd = sem_open(SEM_READ, O_CREAT, 0666, 0);

        if (shm_obj.sem_rd == SEM_FAILED) {

            munmap(shm_obj.shm_ptr, SHM_SIZE);

            shm_unlink(SHM_NAME);

            sem_close(shm_obj.sem_wr);

            sem_unlink(SEM_WRITE);

            close(fd);

            const char mes[] = "Error sem_open.\n";

            exitParent(SEM_ERROR, mes, sizeof(mes));

        }


        pid_t pid = fork();

        if (pid == -1) {

            sem_close(shm_obj.sem_rd);

            sem_close(shm_obj.sem_wr);

            sem_unlink(SEM_WRITE);

            sem_unlink(SEM_READ);

            close(fd);

            munmap(shm_obj.shm_ptr, SHM_SIZE);

            shm_unlink(SHM_NAME);

            const char mes[] = "Error: failed to create process\n";

            exitParent(CREATE_PROCESS_ERR, mes, sizeof(mes));

        }

        else if (pid == 0) {


            char path[PATH_SIZE];

            if (snprintf(path, sizeof(path) - sizeof(char), "%s/%s", progpath,
CHILDE_PROGRAM_NAME) < 0){
```

```
                const char mes[] = "Error: failed to create full name of childe process\n";
                write(STDERR_FILENO, mes, sizeof(mes));
                sem_post(shm_obj.sem_wr);
                exit(CAT_ERR);
            }
            char fd_str[FILE_NAME_BUF_SIZE];
            if (snprintf(fd_str, sizeof(fd_str), "%d", fd) < 0) {
                const char mes[] = "Error: failed to convert int FD to str FD\n";
                write(STDERR_FILENO, mes, sizeof(mes));
                sem_post(shm_obj.sem_wr);
                exit(CONVERT_ERR);
            }

            char* const argv[] = {CHILDE_PROGRAM_NAME, fd_str, NULL};

            int status = execv(path, argv);

            if (status == -1) {
                char mes[] = "Error: failed to exec into new exectuable image\n";
                write(STDERR_FILENO, mes, sizeof(mes));
                sem_post(shm_obj.sem_wr);
                exit(EXEC_ERR);
            }
        }
        // parent
        else {
            close(fd);
            char strBuf[STR_BUF_SIZE];
            strBuf[0] = (char)READY_TO_WRITE;
            strBuf[1] = '\0';
            memcpy(shm_obj.shm_ptr, strBuf, STR_BUF_SIZE);
            while (1) {
                sem_post(shm_obj.sem_rd);
                sem_wait(shm_obj.sem_wr);
                memcpy(strBuf, shm_obj.shm_ptr, STR_BUF_SIZE - 1);
                if (strBuf[0] == (char)READY_TO_READ) {
                    int number = read(STDIN_FILENO, strBuf, (STR_BUF_SIZE - 1) *
sizeof(char));
                    strBuf[STR_BUF_SIZE - 1] = '\0';
                    if (number == -1) {
                        strBuf[0] = (char)ERROR_MES_FROM_PARENT;
                        strBuf[1] = '\0';
                        memcpy(shm_obj.shm_ptr, strBuf, STR_BUF_SIZE);
                        sem_post(shm_obj.sem_rd);
                        wait(NULL);
                        sem_close(shm_obj.sem_rd);
                        sem_close(shm_obj.sem_wr);
                        sem_unlink(SEM_WRITE);
                        sem_unlink(SEM_READ);
                        munmap(shm_obj.shm_ptr, SHM_SIZE);
                        shm_unlink(SHM_NAME);
                        const char mes[] = "Error: failed to read from STDIN\n";
                        exitParent(READ_ERR, mes, sizeof(mes));
                    }
                    strBuf[number] = '\0';
                    memcpy(shm_obj.shm_ptr, strBuf, STR_BUF_SIZE);
                    sem_post(shm_obj.sem_rd);
                }
                else if (strBuf[0] == (char)STOP_MES) {
                    wait(NULL);
                    sem_close(shm_obj.sem_rd);
                    sem_close(shm_obj.sem_wr);
                    sem_unlink(SEM_WRITE);
                    sem_unlink(SEM_READ);
                    munmap(shm_obj.shm_ptr, SHM_SIZE);
                    shm_unlink(SHM_NAME);
                    exit(EXIT_SUCCESS);
                }
                else {
                    const char mes[] = "Error: error in child process\n";
                    int status = 0;
                    sem_post(shm_obj.sem_rd);
                    wait(&status);
                    sem_close(shm_obj.sem_rd);
                    sem_close(shm_obj.sem_wr);
                    sem_unlink(SEM_WRITE);
                    sem_unlink(SEM_READ);
                    munmap(shm_obj.shm_ptr, SHM_SIZE);
                    shm_unlink(SHM_NAME);
                    exitParent(CHILD_ERROR + status, mes, sizeof(mes));
                }
            }
        }
    }

    void exitParent(int code, const char* const message, int mes_size) {
        write(STDERR_FILENO, message, mes_size);
        exit(code);
    }
```

childe.c

```
#include <unistd.h>
#include <sys/types.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <ctype.h>
#include <sys/wait.h>
```

```
#include <stdio.h>
#include <string.h>
#include <semaphore.h>
#include <sys/mman.h>
#include "lab3_defines.h"


typedef enum PARSER_FLAGS {
    CARRY_DIVIDED = 0b1,
    CARRY_DIVISOR = 0b10,
    SPACE_FOUND   = 0b100,
    ENDLINE_FOUND = 0b1000,
    CUR_VAL_DIGIT = 0b10000,
    CUR_VAL_SPACE = 0b100000,
    CUR_VAL_ENDL  = 0b1000000,
    CUR_VAL_SIGN  = 0b10000000,
    SIGN_FOUND    = 0b100000000,
    CUR_VAL_ZERO  = 0b1000000000,
    ZERO_FOUND    = 0b10000000000,
} PARSER_FLAGS;

void exitChilde(int, int, int, const char *, int, struct shm_handler*);
void stopChilde(float, int, int, struct shm_handler*);

int main(int argc,char **argv) {


        int fd = atoi(argv[1]);
        char strBuf[STR_BUF_SIZE];
        float divided = 0;
        int divisor = 0;
        int parser_flag = CARRY_DIVIDED | ENDLINE_FOUND;
```

```
            struct shm_handler shm_obj;
            shm_obj.shm_fd = shm_open(SHM_NAME, O_CREAT | O_RDWR, 0666);
            if (shm_obj.shm_fd == -1) {
                close(fd);
                const char mes[] = "Error shm_open.\n";
                exit(SHM_ERROR);
            }

            shm_obj.shm_ptr = (char*)mmap(0, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm_obj.shm_fd, 0);
            if (shm_obj.shm_ptr == MAP_FAILED) {
                close(fd);
                const char mes[] = "Error mmap.\n";
                exit(SHM_ERROR);
            }

            shm_obj.sem_wr = sem_open(SEM_WRITE, O_CREAT, 0666, 0);
            shm_obj.sem_rd = sem_open(SEM_READ, O_CREAT, 0666, 0);
            if (shm_obj.sem_wr == SEM_FAILED || shm_obj.sem_rd == SEM_FAILED) {
                munmap(shm_obj.shm_ptr, SHM_SIZE);
                close(fd);
                const char mes[] = "Error sem_open.\n";
                exit(SEM_ERROR);
            }


            while (1) {

                int number = 0;
                char sig[1] = {(char)READY_TO_READ};
                strBuf[0] = (char)READY_TO_READ;
                strBuf[1] = '\0';
                sem_wait(shm_obj.sem_rd);
                memcpy(shm_obj.shm_ptr, strBuf, STR_BUF_SIZE);


                sem_post(shm_obj.sem_wr);
                sem_wait(shm_obj.sem_rd);
                memcpy(strBuf, shm_obj.shm_ptr, STR_BUF_SIZE);
                number = strlen(strBuf);
                if (number < 1) {
                    const char mes[] = "Error: failed to read from shm\n";
                    exitChilde(READ_ERR, fd, ERROR_MES_FROM_CHILD, mes, sizeof(mes),
&shm_obj);
                }
                char* ptr = strBuf;
                for (; ptr < strBuf + number; ++ptr) {
                    if (*ptr == '0' && (parser_flag & (ENDLINE_FOUND | SPACE_FOUND |
SIGN_FOUND | ZERO_FOUND)))
                        parser_flag |= CUR_VAL_ZERO;
                    else if (isdigit(*ptr)) parser_flag |= CUR_VAL_DIGIT;
                    else if (*ptr == ' ') parser_flag |= CUR_VAL_SPACE;
                    else if (*ptr == '\n') parser_flag |= CUR_VAL_ENDL;
                    else if (*ptr == '-' || *ptr == '+') parser_flag |= CUR_VAL_SIGN;

                    switch (parser_flag) {

                        case CARRY_DIVIDED | CUR_VAL_DIGIT:
                        case CARRY_DIVIDED | CUR_VAL_DIGIT | ENDLINE_FOUND:
                            divided = divided * 10 + *ptr - '0';
                            parser_flag = CARRY_DIVIDED;
                            break;
                        case CARRY_DIVIDED | CUR_VAL_DIGIT | SIGN_FOUND:
                            divided = divided * (*ptr - '0');
                            parser_flag = CARRY_DIVIDED;
                            break;
                        case CARRY_DIVIDED | CUR_VAL_SPACE:
                        case CARRY_DIVIDED | CUR_VAL_SPACE | ZERO_FOUND:
                            parser_flag = CARRY_DIVISOR | SPACE_FOUND;
                            break;

                        case CARRY_DIVIDED | CUR_VAL_ENDL:
                        case CARRY_DIVIDED | CUR_VAL_ENDL | ZERO_FOUND:
                            char result[14];
                            int len = sprintf(result, "%g\n", divided);
                            if (write(fd, result, len * sizeof(char)) == -1) {
                                const char mes[] = "Error: failed to write to file\n";
                                exitChilde(WRITE_ERR, fd, ERROR_MES_FROM_CHILD, mes, sizeof(mes), &shm_obj);
                            }
                            divided = 0;
                            parser_flag = CARRY_DIVIDED | ENDLINE_FOUND;
                            break;

                        case CARRY_DIVIDED | CUR_VAL_SIGN | ENDLINE_FOUND:
                            if (*ptr == '-') divided = -1;
                            else divided = 1;
                            parser_flag = CARRY_DIVIDED | SIGN_FOUND;
                            break;

                        case CARRY_DIVIDED | CUR_VAL_ZERO | ENDLINE_FOUND:
                        case CARRY_DIVIDED | CUR_VAL_ZERO | SIGN_FOUND:
                            divided = 0;
                            parser_flag = CARRY_DIVIDED | ZERO_FOUND;
                            break;
```

```
case CARRY_DIVISOR | CUR_VAL_DIGIT:
case CARRY_DIVISOR | CUR_VAL_DIGIT | SPACE_FOUND:
    divisor = divisor * 10 + *ptr - '0';
    parser_flag = CARRY_DIVISOR;
    break;
case CARRY_DIVISOR | CUR_VAL_DIGIT | SIGN_FOUND:
    divisor = divisor * (*ptr - '0');
    parser_flag = CARRY_DIVISOR;
    break;


case CARRY_DIVISOR | CUR_VAL_SPACE:
case CARRY_DIVISOR | CUR_VAL_SPACE | ZERO_FOUND:
    if (divisor == 0) {
        stopChilde(divided, fd, STOP_MES, &shm_obj);
        divided = 0;
    }
    else {
        divided /= divisor;
        divisor = 0;
        parser_flag = CARRY_DIVISOR | SPACE_FOUND;
    }
    break;


case CARRY_DIVISOR | CUR_VAL_ENDL:
case CARRY_DIVISOR | CUR_VAL_ENDL | ZERO_FOUND:
    if (divisor == 0) {
        stopChilde(divided, fd, STOP_MES, &shm_obj);
        divided = 0;
```

```
                            }
                            else {
                                divided /= divisor;
                                divisor = 0;
                                char result[14];
                                int len = sprintf(result, "%g\n", divided);
                                if (write(fd, result, sizeof(char) * len) == -1) {
                                    const char mes[] = "Error: failed to write to file\n";
                                    exitChilde(WRITE_ERR, fd, ERROR_MES_FROM_CHILD, mes,
sizeof(mes), &shm_obj);
                                }
                                divided = 0;
                                parser_flag = CARRY_DIVIDED | ENDLINE_FOUND;
                            }
                            break;


                        case CARRY_DIVISOR | CUR_VAL_SIGN | SPACE_FOUND:
                            if (*ptr == '-') divisor = -1;
                            else divisor = 1;
                            parser_flag = CARRY_DIVISOR | SIGN_FOUND;
                            break;


                        case CARRY_DIVISOR | CUR_VAL_ZERO | SPACE_FOUND:
                        case CARRY_DIVISOR | CUR_VAL_ZERO | SIGN_FOUND:
                            divisor = 0;
                            parser_flag = CARRY_DIVISOR | ZERO_FOUND;
                            break;

                        default:
                        /*
                            ERRORS:
                             case CARRY_DIVIDED | CUR_VAL_SPACE | ENDLINE_FOUND
                             case CARRY_DIVIDED | CUR_VAL_ENDL | ENDLINE_FOUND
                             case CARRY_DIVISOR | CUR_VAL_SPACE | SPACE_FOUND
                             case CARRY_DIVISOR | CUR_VAL_ENDL | SPACE_FOUND
                             case CARRY_DIVIDED | CUR_VAL_SIGN ...
                             IMPOSSIBLE:
                             case CARRY_DIVIDED | CUR_VAL_DIGIT | SPACE_FOUND : SPACE_FOUND
-> CARRY_DIVISOR
                             case CARRY_DIVIDED | CUR_VAL_SPACE | SPACE_FOUND
                             case CARRY_DIVIDED | CUR_VAL_ENDL | SPACE_FOUND
                             case CARRY_DIVISOR | CUR_VAL_DIGIT | ENDLINE_FOUND :
ENDLINE_FOUND -> CARRY_DIVIDED
                             case CARRY_DIVISOR | CUR_VAL_SPACE | ENDLINE_FOUND
                             case CARRY_DIVISOR | CUR_VAL_SPACE | ENDLINE_FOUND
                             case CARRY_DIVIDED | CUR_VAL_SIGN | SPACE_FOUND ...
                         */
                             const char mes[] = "Error: wrong inscturction\n";
                             exitChilde(WRONG_INSTRUCTION_ERR, fd, ERROR_MES_FROM_CHILD,
mes, sizeof(mes), &shm_obj);
                             break;
                     }
                 }
             }
     }

     void exitChilde(int code, int fd, int sig, const char *const mes, int mes_size, struct
shm_handler *shm_obj) {
         char buf[2] = {(char)sig, '\0'};
         // shm_obj.sem_wr
         sem_wait(shm_obj->sem_rd);
         memcpy(shm_obj->shm_ptr, buf, sizeof(buf));
         write(STDERR_FILENO, mes, mes_size);
         close(fd);
         sem_post(shm_obj->sem_wr);
         sem_close(shm_obj->sem_rd);
```

```
    sem_close(shm_obj->sem_wr);
    exit(code);
}

void stopChilde(float divided, int fd, int mes_to_parent, struct shm_handler *shm_obj) {
    char res[14];
    int len = sprintf(res, "%g\n", divided);
    if (write(fd, res, sizeof(char) * len) == -1) {
        const char mes[] = "Error: failed to write to file\n";
        exitChilde(WRITE_ERR, fd, ERROR_MES_FROM_CHILD, mes, sizeof(mes), shm_obj);
    }
    res[0] = (char)mes_to_parent;
    res[1] = '\0';
    sem_wait(shm_obj->sem_rd);
    memcpy(shm_obj->shm_ptr, res, sizeof(res));
    close(fd);
    sem_post(shm_obj->sem_wr);
    sem_close(shm_obj->sem_rd);
    sem_close(shm_obj->sem_wr);
    exit(EXIT_SUCCESS);
}
```

Протокол работы программы

Strace:

```
26017 execve("./parent", ["./parent"], 0x7ffd5c3dcd58 /* 61 vars */) = 0
26017 brk(NULL)                         = 0x55ff377ac000
26017 mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fd561db5000
26017 access("/etc/ld.so.preload", R_OK) = -1 ENOENT (Нет такого файла или каталога)
26017 openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
26017 newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=92170, ...}, AT_EMPTY_PATH) = 0
26017 mmap(NULL, 92170, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fd561d9e000
```

```
     26017 close(3)                          = 0

     26017 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3

     26017 read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\20t\2\0\0\0\0\0"...,
832) = 832

     26017 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

     26017 newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=1922136, ...}, AT_EMPTY_PATH) =
0

     26017 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

     26017 mmap(NULL, 1970000, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fd561bbd000

     26017 mmap(0x7fd561be3000, 1396736, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x26000) = 0x7fd561be3000

     26017 mmap(0x7fd561d38000, 339968, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x17b000) = 0x7fd561d38000

     26017 mmap(0x7fd561d8b000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1ce000) = 0x7fd561d8b000

     26017 mmap(0x7fd561d91000, 53072, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fd561d91000

     26017 close(3)                          = 0

     26017 mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fd561bba000

     26017 arch_prctl(ARCH_SET_FS, 0x7fd561bba740) = 0

     26017 set_tid_address(0x7fd561bbaa10)   = 26017

     26017 set_robust_list(0x7fd561bbaa20, 24) = 0

     26017 rseq(0x7fd561bbb060, 0x20, 0, 0x53053053) = 0

     26017 mprotect(0x7fd561d8b000, 16384, PROT_READ) = 0

     26017 mprotect(0x55ff361ca000, 4096, PROT_READ) = 0

     26017 mprotect(0x7fd561de7000, 8192, PROT_READ) = 0

     26017 prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY})
= 0

     26017 munmap(0x7fd561d9e000, 92170)     = 0

     26017 readlink("/proc/self/exe", "/home/vnadez/Projects/OS_LABS/LA"..., 1023) = 42

     26017 read(0, "out.txt\n", 15)          = 8

     26017 openat(AT_FDCWD, "out.txt", O_WRONLY|O_CREAT|O_TRUNC, 0707) = 3

     26017 openat(AT_FDCWD, "/dev/shm/shared_memory", O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC,
0666) = 6

     26017 ftruncate(6, 8192)                = 0

     26017 mmap(NULL, 500, PROT_READ|PROT_WRITE, MAP_SHARED, 6, 0) = 0x7fd561db4000

     26017 openat(AT_FDCWD, "/dev/shm/sem.sem_write_", O_RDWR|O_NOFOLLOW) = -1 ENOENT (Нет
такого файла или каталога)
```

```
26017 getrandom("\xc3\x2c\x14\xde\xc5\xb3\x58\x58", 8, GRND_NONBLOCK) = 8

26017 newfstatat(AT_FDCWD, "/dev/shm/sem.dKmxpf", 0x7ffc27246ca0, AT_SYMLINK_NOFOLLOW) = -1 ENOENT (Нет такого файла или каталога)

26017 openat(AT_FDCWD, "/dev/shm/sem.dKmxpf", O_RDWR|O_CREAT|O_EXCL, 0666) = 7

26017 write(7, "\0\0\0\0\0\0\0\0\200\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0", 32) = 32

26017 mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 7, 0) = 0x7fd561db3000

26017 link("/dev/shm/sem.dKmxpf", "/dev/shm/sem.sem_write_") = 0

26017 newfstatat(7, "", {st_mode=S_IFREG|0644, st_size=32, ...}, AT_EMPTY_PATH) = 0

26017 getrandom("\x49\xb4\x63\xa9\xa9\x55\x1a\x00", 8, GRND_NONBLOCK) = 8

26017 brk(NULL)                         = 0x55ff377ac000

26017 brk(0x55ff377cd000)               = 0x55ff377cd000

26017 unlink("/dev/shm/sem.dKmxpf")     = 0

26017 close(7)                          = 0

26017 openat(AT_FDCWD, "/dev/shm/sem.sem_read_", O_RDWR|O_NOFOLLOW) = -1 ENOENT (Нет такого файла или каталога)

26017 getrandom("\x25\xcb\x21\xe7\x2f\x01\x5a\xa3", 8, GRND_NONBLOCK) = 8

26017 newfstatat(AT_FDCWD, "/dev/shm/sem.3TWpWG", 0x7ffc27246ca0, AT_SYMLINK_NOFOLLOW) = -1 ENOENT (Нет такого файла или каталога)

26017 openat(AT_FDCWD, "/dev/shm/sem.3TWpWG", O_RDWR|O_CREAT|O_EXCL, 0666) = 7

26017 write(7, "\0\0\0\0\0\0\0\0\200\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0", 32) = 32

26017 mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 7, 0) = 0x7fd561db2000

26017 link("/dev/shm/sem.3TWpWG", "/dev/shm/sem.sem_read_") = 0

26017 newfstatat(7, "", {st_mode=S_IFREG|0644, st_size=32, ...}, AT_EMPTY_PATH) = 0

26017 unlink("/dev/shm/sem.3TWpWG")     = 0

26017 close(7)                          = 0

26017 clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7fd561bbaa10) = 26020

26017 close(3 <unfinished ...>

26020 set_robust_list(0x7fd561bbaa20, 24 <unfinished ...>

26017 <... close resumed>)              = 0

26017 futex(0x7fd561db3000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL, FUTEX_BITSET_MATCH_ANY <unfinished ...>

26020 <... set_robust_list resumed>)    = 0

26020 execve("/home/vnadez/Projects/OS_LABS/LAB_3/childe", ["childe", "3"], 0x7ffc272477c8 /* 61 vars */) = 0

26020 brk(NULL)                         = 0x556ecec42000

26020 mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f1dde853000
```

```
26020 access("/etc/ld.so.preload", R_OK) = -1 ENOENT (Нет такого файла или каталога)
26020 openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 6
26020 newfstatat(6, "", {st_mode=S_IFREG|0644, st_size=92170, ...}, AT_EMPTY_PATH) = 0
26020 mmap(NULL, 92170, PROT_READ, MAP_PRIVATE, 6, 0) = 0x7f1dde83c000
26020 close(6)                          = 0
26020 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 6
26020 read(6, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\20t\2\0\0\0\0\0"..., 832) = 832
26020 pread64(6, "\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
26020 newfstatat(6, "", {st_mode=S_IFREG|0755, st_size=1922136, ...}, AT_EMPTY_PATH) = 0
26020 pread64(6, "\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
26020 mmap(NULL, 1970000, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 6, 0) = 0x7f1dde65b000
26020 mmap(0x7f1dde681000, 1396736, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 6, 0x26000) = 0x7f1dde681000
26020 mmap(0x7f1dde7d6000, 339968, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 6, 0x17b000) = 0x7f1dde7d6000
26020 mmap(0x7f1dde829000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 6, 0x1ce000) = 0x7f1dde829000
26020 mmap(0x7f1dde82f000, 53072, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f1dde82f000
26020 close(6)                          = 0
26020 mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f1dde658000
26020 arch_prctl(ARCH_SET_FS, 0x7f1dde658740) = 0
26020 set_tid_address(0x7f1dde658a10)   = 26020
26020 set_robust_list(0x7f1dde658a20, 24) = 0
26020 rseq(0x7f1dde659060, 0x20, 0, 0x53053053) = 0
26020 mprotect(0x7f1dde829000, 16384, PROT_READ) = 0
26020 mprotect(0x556ece5ae000, 4096, PROT_READ) = 0
26020 mprotect(0x7f1dde885000, 8192, PROT_READ) = 0
26020 prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
26020 munmap(0x7f1dde83c000, 92170)     = 0
26020 openat(AT_FDCWD, "/dev/shm/shared_memory", O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC, 0666) = 6
26020 mmap(NULL, 500, PROT_READ|PROT_WRITE, MAP_SHARED, 6, 0) = 0x7f1dde852000
26020 openat(AT_FDCWD, "/dev/shm/sem.sem_write_", O_RDWR|O_NOFOLLOW) = 7
```

```
26020 newfstatat(7, "", {st_mode=S_IFREG|0644, st_size=32, ...}, AT_EMPTY_PATH) = 0
26020 getrandom("\xd8\x15\x57\xa4\xa8\xb8\x6b\x2f", 8, GRND_NONBLOCK) = 8
26020 brk(NULL)                         = 0x556ecec42000
26020 brk(0x556ecec63000)               = 0x556ecec63000
26020 mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 7, 0) = 0x7f1dde851000
26020 close(7)                          = 0
26020 openat(AT_FDCWD, "/dev/shm/sem.sem_read_", O_RDWR|O_NOFOLLOW) = 7
26020 newfstatat(7, "", {st_mode=S_IFREG|0644, st_size=32, ...}, AT_EMPTY_PATH) = 0
26020 mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 7, 0) = 0x7f1dde850000
26020 close(7)                          = 0
26020 futex(0x7f1dde851000, FUTEX_WAKE, 1 <unfinished ...>
26017 <... futex resumed>)              = 0
26020 <... futex resumed>)              = 1
26017 read(0,  <unfinished ...>
26020 futex(0x7f1dde850000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL,
FUTEX_BITSET_MATCH_ANY <unfinished ...>
26017 <... read resumed>"12 2 3\n", 499) = 7
26017 futex(0x7fd561db2000, FUTEX_WAKE, 1) = 1
26020 <... futex resumed>)              = 0
26017 futex(0x7fd561db2000, FUTEX_WAKE, 1 <unfinished ...>
26020 write(3, "2\n", 2 <unfinished ...>
26017 <... futex resumed>)              = 0
26020 <... write resumed>)              = 2
26020 futex(0x7f1dde850000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL,
FUTEX_BITSET_MATCH_ANY <unfinished ...>
26017 read(0, "12 4\n", 499)            = 5
26017 futex(0x7fd561db2000, FUTEX_WAKE, 1) = 1
26020 <... futex resumed>)              = 0
26017 futex(0x7fd561db2000, FUTEX_WAKE, 1 <unfinished ...>
26020 write(3, "3\n", 2 <unfinished ...>
26017 <... futex resumed>)              = 0
26020 <... write resumed>)              = 2
26017 futex(0x7fd561db3000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL,
FUTEX_BITSET_MATCH_ANY <unfinished ...>
26020 futex(0x7f1dde851000, FUTEX_WAKE, 1 <unfinished ...>
26017 <... futex resumed>)              = -1 EAGAIN (Ресурс временно недоступен)
26020 <... futex resumed>)              = 0
```

```
     26017 read(0,  <unfinished ...>

     26020 futex(0x7f1dde850000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL,
FUTEX_BITSET_MATCH_ANY <unfinished ...>

     26017 <... read resumed>"100 -10 5\n", 499) = 10

     26017 futex(0x7fd561db2000, FUTEX_WAKE, 1) = 1

     26020 <... futex resumed>)              = 0

     26017 futex(0x7fd561db2000, FUTEX_WAKE, 1) = 0

     26017 futex(0x7fd561db3000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL,
FUTEX_BITSET_MATCH_ANY <unfinished ...>

     26020 write(3, "-2\n", 3)               = 3

     26020 futex(0x7f1dde851000, FUTEX_WAKE, 1 <unfinished ...>

     26017 <... futex resumed>)              = 0

     26020 <... futex resumed>)              = 1

     26017 read(0,  <unfinished ...>

     26020 futex(0x7f1dde850000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL,
FUTEX_BITSET_MATCH_ANY <unfinished ...>

     26017 <... read resumed>"124 5\n", 499) = 6

     26017 futex(0x7fd561db2000, FUTEX_WAKE, 1) = 1

     26020 <... futex resumed>)              = 0

     26017 futex(0x7fd561db2000, FUTEX_WAKE, 1 <unfinished ...>

     26020 write(3, "24.8\n", 5 <unfinished ...>

     26017 <... futex resumed>)              = 0

     26017 futex(0x7fd561db3000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL,
FUTEX_BITSET_MATCH_ANY <unfinished ...>

     26020 <... write resumed>)              = 5

     26020 futex(0x7f1dde851000, FUTEX_WAKE, 1 <unfinished ...>

     26017 <... futex resumed>)              = 0

     26020 <... futex resumed>)              = 1

     26017 read(0,  <unfinished ...>

     26020 futex(0x7f1dde850000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL,
FUTEX_BITSET_MATCH_ANY <unfinished ...>

     26017 <... read resumed>"0\n", 499)     = 2

     26017 futex(0x7fd561db2000, FUTEX_WAKE, 1) = 1

     26020 <... futex resumed>)              = 0

     26017 futex(0x7fd561db3000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL,
FUTEX_BITSET_MATCH_ANY <unfinished ...>

     26020 write(3, "0\n", 2)                = 2

     26020 futex(0x7f1dde851000, FUTEX_WAKE, 1 <unfinished ...>
     26017 <... futex resumed>)             = 0
     26020 <... futex resumed>)             = 1
     26017 read(0,  <unfinished ...>
     26020 futex(0x7f1dde850000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL, FUTEX_BITSET_MATCH_ANY <unfinished ...>
     26017 <... read resumed>"45 0\n", 499)  = 5
     26017 futex(0x7fd561db2000, FUTEX_WAKE, 1) = 1
     26020 <... futex resumed>)             = 0
     26017 futex(0x7fd561db2000, FUTEX_WAKE, 1 <unfinished ...>
     26020 write(3, "45\n", 3 <unfinished ...>
     26017 <... futex resumed>)             = 0
     26020 <... write resumed>)             = 3
     26017 futex(0x7fd561db3000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL, FUTEX_BITSET_MATCH_ANY <unfinished ...>
     26020 close(3)                         = 0
     26020 futex(0x7f1dde851000, FUTEX_WAKE, 1) = 1
     26017 <... futex resumed>)             = 0
     26020 munmap(0x7f1dde850000, 32 <unfinished ...>
     26017 wait4(-1,  <unfinished ...>
     26020 <... munmap resumed>)            = 0
     26020 munmap(0x7f1dde851000, 32)       = 0
     26020 exit_group(0)                    = ?
     26020 +++ exited with 0 +++
     26017 <... wait4 resumed>NULL, 0, NULL) = 26020
     26017 --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=26020, si_uid=1000, si_status=0, si_utime=0, si_stime=0} ---
     26017 munmap(0x7fd561db2000, 32)       = 0
     26017 munmap(0x7fd561db3000, 32)       = 0
     26017 unlink("/dev/shm/sem.sem_write_") = 0
     26017 unlink("/dev/shm/sem.sem_read_")  = 0
     26017 munmap(0x7fd561db4000, 500)      = 0
     26017 unlink("/dev/shm/shared_memory")  = 0
     26017 exit_group(0)                    = ?
     26017 +++ exited with 0 +++
```

Вывод

При выполнении этой работы я получил практические навыки в создании и управлении разделяемой памятью, а так же в использовании средств блокировок.