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-214Б-23

Студент: Сетраков Ф.С.

Преподаватель: Бахарев В.Д. (ФИИТ)

Оценка: _____

Дата: 24.12.24

Москва, 2024

Постановка задачи

Вариант 22.

Необходимо составить и отладить программу на Си. Родительский процесс создаёт два дочерних процесса. Первой строкой пользователь вводит имя файла, которое будет использовано для открытия File с таким именем на запись для child1. Аналогично для второй строки и процесса child2. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их с вероятностью 80% в child1, иначе в child2. Дочерние процессы инвертируют строки.

Общий метод и алгоритм решения

Использованные системные вызовы:

- shm_open - открытие shared memory
- sem_open - открытие семафора
- mmap - маппинг памяти
- munmap - удаляет маппинг
- fork - создание дочернего процесса
- waitpid - ожидание завершения процессов
- sem_wait - “блокировка” семафора
- sem_post - “разблокировка” семафора

Затем шла работа над библиотекой multitasking. Вся суть заключается в том, чтобы считать имя файлов, в которые будет выводиться результат, затем считывать строки, инвертировать и отправлять в один из дочерних процессов.

Код программы

parent.cpp

```
#include <cstring>
#include "parent.h"

#define SHM_SIZE 4096
#define SEM_PARENT_READY_1 "/sem_parent_ready_1"
#define SEM_CHILD_READY_1 "/sem_child_ready_1"
#define SEM_PARENT_READY_2 "/sem_parent_ready_2"
#define SEM_CHILD_READY_2 "/sem_child_ready_2"

error parent_loop(char* shm1, char* shm2, sem_t* sem_parent_ready_1, sem_t* sem_child_ready_1,
                  sem_t* sem_parent_ready_2, sem_t* sem_child_ready_2) {
        std::string buffer;
        int bytes = std_in(&buffer);
        char* target_shm;
        sem_t* target_sem_parent_ready;
        sem_t* target_sem_child_ready;

        while (bytes > 0 && buffer != "STOP\n") {
        if (std::rand() % 100 >= 80) {
        target_shm = shm2;
        target_sem_parent_ready = sem_parent_ready_2;
        target_sem_child_ready = sem_child_ready_2;
        } else {
        target_shm = shm1;
        target_sem_parent_ready = sem_parent_ready_1;
        target_sem_child_ready = sem_child_ready_1;
```

```
        }

        sem_wait(target_sem_child_ready);
        strncpy(target_shm, buffer.c_str(), SHM_SIZE);
        sem_post(target_sem_parent_ready);

        bytes = std_in(&buffer);
        }

        sem_wait(sem_child_ready_1);
        strncpy(shm1, "STOP\n", SHM_SIZE);
        sem_post(sem_parent_ready_1);

        sem_wait(sem_child_ready_2);
        strncpy(shm2, "STOP\n", SHM_SIZE);
        sem_post(sem_parent_ready_2);

        return STATE_OK;
}

error parent_process() {
        std::srand(std::time(nullptr));

        std::string file1_name, file2_name;
        if (file_scan(STDIN_FILENO, &file1_name) <= 1 || file_scan(STDIN_FILENO, &file2_name) <=
1 ||
        file1_name == file2_name) {
        return ERROR_INVALID_INPUT;
        }
        file1_name.erase(file1_name.length() - 1);
        file2_name.erase(file2_name.length() - 1);

        // Создаём shared memory
        int shm_fd1 = shm_open("/shm1", O_CREAT | O_RDWR, 0666);
        int shm_fd2 = shm_open("/shm2", O_CREAT | O_RDWR, 0666);
        if (shm_fd1 == -1 || shm_fd2 == -1 || ftruncate(shm_fd1, SHM_SIZE) == -1 || ftruncate(shm_fd2,
SHM_SIZE) == -1) {
        return ERROR_PIPE_FAILED;
        }

        char* shm1 = static_cast<char*>(mmap(nullptr, SHM_SIZE, PROT_READ | PROT_WRITE,
MAP_SHARED, shm_fd1, 0));
        char* shm2 = static_cast<char*>(mmap(nullptr, SHM_SIZE, PROT_READ | PROT_WRITE,
MAP_SHARED, shm_fd2, 0));
        if (shm1 == MAP_FAILED || shm2 == MAP_FAILED) {
        return ERROR_PIPE_FAILED;
        }

        sem_t* sem_parent_ready_1 = sem_open(SEM_PARENT_READY_1, O_CREAT, 0666, 0);
        sem_t* sem_child_ready_1 = sem_open(SEM_CHILD_READY_1, O_CREAT, 0666, 1);
        sem_t* sem_parent_ready_2 = sem_open(SEM_PARENT_READY_2, O_CREAT, 0666, 0);
        sem_t* sem_child_ready_2 = sem_open(SEM_CHILD_READY_2, O_CREAT, 0666, 1);

        if (sem_parent_ready_1 == SEM_FAILED || sem_child_ready_1 == SEM_FAILED ||
        sem_parent_ready_2 == SEM_FAILED || sem_child_ready_2 == SEM_FAILED) {
        return ERROR_PIPE_FAILED;
        }
```

```
	pid_t pid1 = fork();
	if (pid1 == 0) {
	child_process(shm1, sem_parent_ready_1, sem_child_ready_1, file1_name);
	}

	pid_t pid2 = fork();
	if (pid2 == 0) {
	child_process(shm2, sem_parent_ready_2, sem_child_ready_2, file2_name);
	}

	error err = parent_loop(shm1, shm2, sem_parent_ready_1, sem_child_ready_1,
sem_parent_ready_2, sem_child_ready_2);

	waitpid(pid1, nullptr, 0);
	waitpid(pid2, nullptr, 0);

	munmap(shm1, SHM_SIZE);
	munmap(shm2, SHM_SIZE);
	shm_unlink("/shm1");
	shm_unlink("/shm2");
	sem_close(sem_parent_ready_1);
	sem_close(sem_child_ready_1);
	sem_close(sem_parent_ready_2);
	sem_close(sem_child_ready_2);
	sem_unlink(SEM_PARENT_READY_1);
	sem_unlink(SEM_CHILD_READY_1);
	sem_unlink(SEM_PARENT_READY_2);
	sem_unlink(SEM_CHILD_READY_2);

	return err;
}
```

child.cpp

```
#include "child.h"

void invert_string(std::string* s) {
	int p1 = 0, p2 = (*s).size() - 1;
	while (p1 < p2) {
	std::swap((*s)[p1], (*s)[p2]);
	p1++;
	p2--;
	}
}

sem_t child_process(char* shm, sem_t* sem_parent_ready, sem_t* sem_child_ready, const std::string&
file_name) {
	int write_fd = open(file_name.c_str(), O_WRONLY | O_CREAT | O_APPEND, S_IRUSR |
S_IWUSR);
	if (write_fd == -1) {
	log_stderr("Error opening file in child process");
	exit(1);
	}

	std::string buffer;

	while (true) {
	sem_wait(sem_parent_ready);
	buffer = std::string(shm);
```

```
        if (buffer == "STOP\n") {
        break;
        }

        buffer.erase(buffer.size() - 1);
        invert_string(&buffer);

        if (file_print(write_fd, buffer + "\n") == -1) {
        log_stderr("Error writing to file");
        exit(1);
        }

        sem_post(sem_child_ready);
        }

        close(write_fd);
        exit(0);
}
```

io.cpp

```
#include "io.h"

int std_in(std::string* message) {
        return file_scan(STDIN_FILENO, message);
}

void std_out(const std::string& message) {
        file_print(STDOUT_FILENO, message);
}

void log_stderr(const std::string& message) {
        file_print(STDERR_FILENO, message);
}

int file_scan(int input_file, std::string* message) {
        char buffer[DEFAULT_BUFF_SIZE];
        std::string output;
        ssize_t bytes = read(input_file, buffer, DEFAULT_BUFF_SIZE);
        for (int i = 0; i < bytes; i++) {
        output += buffer[i];
        }
        *message = output;
        return (int) bytes;
}

int file_print(int output_file, const std::string& message) {
        ssize_t bytes = write(output_file, message.c_str(), message.size());
        return (int) bytes;
}
```

error_handlers.cpp

```
#include "error_handlers.h"

void log_error(error err) {
        switch (err) {
        case ERROR_STOP_FAILED: {
        std_out("ERROR: CANT STOP ONE OF CHILD PROCESSES\n");
```

```
        break;
        }
        case ERROR_PIPE_FAILED: {
        std_out("ERROR: CANT CREATE PIPES\n");
        break;
        }
        case ERROR_FORK_FAILED: {
        std_out("ERROR: FORK FAILED\n");
        break;
        }
        case ERROR_INVALID_INPUT: {
        std_out("ERROR: INVALID INPUT\n");
        break;
        }
        case ERROR_CANT_WRITE_FILE: {
        std_out("ERROR: CANT WRITE FILE\n");
        break;
        }
        default: {
        std_out("UNKNOWN ERROR CODE\n");
        break;
        }
        }
}
```

main.cpp

```
#include "multitasking/parent.h"
#include "errlib/errors.h"
#include "errlib/error_handlers.h"

int main() {
        error err = parent_process();
        if (err != STATE_OK) {
        log_error(err);
        }
        return err;
}
```

Протокол работы программы

```
setrakovfs@osx ~/oslabs/OSLABS/lab3 $ docker run -it --rm my-cpp-strace-image strace ./a.out
execve("./a.out", ["./a.out"], 0x7ffeee3fcef0 /* 4 vars */) = 0
brk(NULL)                               = 0x5f1f38fc7000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x769354f14000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=9303, ...}) = 0
mmap(NULL, 9303, PROT_READ, MAP_PRIVATE, 3, 0) = 0x769354f11000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libstdc++.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0644, st_size=2592224, ...}) = 0
mmap(NULL, 2609472, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x769354c93000
```

```
mmap(0x769354d30000, 1343488, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x9d000) = 0x769354d30000
mmap(0x769354e78000, 552960, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1e5000) = 0x769354e78000
mmap(0x769354eff000, 57344, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x26b000) = 0x769354eff000
mmap(0x769354f0d000, 12608, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x769354f0d000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libgcc_s.so.1", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0644, st_size=183024, ...}) = 0
mmap(NULL, 185256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x769354c65000
mmap(0x769354c69000, 147456, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x4000) = 0x769354c69000
mmap(0x769354c8d000, 16384, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x28000) = 0x769354c8d000
mmap(0x769354c91000, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2b000) = 0x769354c91000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\243\2\0\0\0\0\0"..., 832) = 832
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x769354a53000
mmap(0x769354a7b000, 1605632, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x769354a7b000
mmap(0x769354c03000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1b0000) = 0x769354c03000
mmap(0x769354c52000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x769354c52000
mmap(0x769354c58000, 52624, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x769354c58000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libm.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0644, st_size=952616, ...}) = 0
mmap(NULL, 950296, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x76935496a000
mmap(0x76935497a000, 520192, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x10000) = 0x76935497a000
mmap(0x7693549f9000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x8f000) = 0x7693549f9000
mmap(0x769354a51000, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0xe7000) = 0x769354a51000
close(3)                                = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x769354968000
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x769354965000
arch_prctl(ARCH_SET_FS, 0x769354965740) = 0
set_tid_address(0x769354965a10)         = 9
set_robust_list(0x769354965a20, 24)     = 0
rseq(0x769354966060, 0x20, 0, 0x53053053) = 0
mprotect(0x769354c52000, 16384, PROT_READ) = 0
mprotect(0x769354a51000, 4096, PROT_READ) = 0
```

```
mprotect(0x769354c91000, 4096, PROT_READ) = 0
mprotect(0x769354eff000, 45056, PROT_READ) = 0
mprotect(0x5f1f38f35000, 4096, PROT_READ) = 0
mprotect(0x769354f4c000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x769354f11000, 9303)            = 0
futex(0x769354f0d7bc, FUTEX_WAKE_PRIVATE, 2147483647) = 0
getrandom("\xcb\xfc\x80\x91\x26\x20\x10\x53", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x5f1f38fc7000
brk(0x5f1f38fe8000)                     = 0x5f1f38fe8000
read(0, file1.txt
"file1.txt\n", 1024)                    = 10
read(0, file2.txt
"file2.txt\n", 1024)                    = 10
openat(AT_FDCWD, "/dev/shm/shm1", O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC, 0666) = 3
openat(AT_FDCWD, "/dev/shm/shm2", O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC, 0666) = 4
ftruncate(3, 4096)                      = 0
ftruncate(4, 4096)                      = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x769354f13000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x769354f12000
openat(AT_FDCWD, "/dev/shm/sem.sem_parent_ready_1", O_RDWR|O_NOFOLLOW|O_CLOEXEC) =
-1 ENOENT (No such file or directory)
getrandom("\x28\x1b\xfe\xc7\x02\x8a\xa4\xe9", 8, GRND_NONBLOCK) = 8
newfstatat(AT_FDCWD, "/dev/shm/sem.SMsg5H", 0x7fffa7ec7010, AT_SYMLINK_NOFOLLOW) = -1
ENOENT (No such file or directory)
openat(AT_FDCWD, "/dev/shm/sem.SMsg5H",
O_RDWR|O_CREAT|O_EXCL|O_NOFOLLOW|O_CLOEXEC, 0666) = 5
write(5, "\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\0", 32) = 32
mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 5, 0) = 0x769354f11000
link("/dev/shm/sem.SMsg5H", "/dev/shm/sem.sem_parent_ready_1") = 0
fstat(5, {st_mode=S_IFREG|0644, st_size=32, ...}) = 0
unlink("/dev/shm/sem.SMsg5H")           = 0
close(5)                                = 0
openat(AT_FDCWD, "/dev/shm/sem.sem_child_ready_1", O_RDWR|O_NOFOLLOW|O_CLOEXEC) =
-1 ENOENT (No such file or directory)
getrandom("\xc4\x08\x16\x55\x65\x4f\x22\x90", 8, GRND_NONBLOCK) = 8
newfstatat(AT_FDCWD, "/dev/shm/sem.kv3n1x", 0x7fffa7ec7010, AT_SYMLINK_NOFOLLOW) = -1
ENOENT (No such file or directory)
openat(AT_FDCWD, "/dev/shm/sem.kv3n1x",
O_RDWR|O_CREAT|O_EXCL|O_NOFOLLOW|O_CLOEXEC, 0666) = 5
write(5, "\1\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\0", 32) = 32
mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 5, 0) = 0x769354964000
link("/dev/shm/sem.kv3n1x", "/dev/shm/sem.sem_child_ready_1") = 0
fstat(5, {st_mode=S_IFREG|0644, st_size=32, ...}) = 0
unlink("/dev/shm/sem.kv3n1x")           = 0
close(5)                                = 0
openat(AT_FDCWD, "/dev/shm/sem.sem_parent_ready_2", O_RDWR|O_NOFOLLOW|O_CLOEXEC) =
-1 ENOENT (No such file or directory)
getrandom("\x44\xcd\xbb\xc2\xba\xeb\x6a\x00", 8, GRND_NONBLOCK) = 8
newfstatat(AT_FDCWD, "/dev/shm/sem.MlRDPV", 0x7fffa7ec7010, AT_SYMLINK_NOFOLLOW) = -1
ENOENT (No such file or directory)
openat(AT_FDCWD, "/dev/shm/sem.MlRDPV",
O_RDWR|O_CREAT|O_EXCL|O_NOFOLLOW|O_CLOEXEC, 0666) = 5
write(5, "\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\0", 32) = 32
mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 5, 0) = 0x769354963000
link("/dev/shm/sem.MlRDPV", "/dev/shm/sem.sem_parent_ready_2") = 0
```

```
fstat(5, {st_mode=S_IFREG|0644, st_size=32, ...}) = 0
unlink("/dev/shm/sem.MlRDPV")           = 0
close(5)                                = 0
openat(AT_FDCWD, "/dev/shm/sem.sem_child_ready_2", O_RDWR|O_NOFOLLOW|O_CLOEXEC) =
-1 ENOENT (No such file or directory)
getrandom("\x00\xca\xc3\x2e\x58\xf7\x1c\xfc", 8, GRND_NONBLOCK) = 8
getrandom("\x26\xa9\x98\x39\xae\x0f\xd9\x94", 8, GRND_NONBLOCK) = 8
newfstatat(AT_FDCWD, "/dev/shm/sem.WnwtPC", 0x7fffa7ec7010, AT_SYMLINK_NOFOLLOW) = -1
ENOENT (No such file or directory)
openat(AT_FDCWD, "/dev/shm/sem.WnwtPC",
O_RDWR|O_CREAT|O_EXCL|O_NOFOLLOW|O_CLOEXEC, 0666) = 5
write(5, "\1\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\0", 32) = 32
mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 5, 0) = 0x769354962000
link("/dev/shm/sem.WnwtPC", "/dev/shm/sem.sem_child_ready_2") = 0
fstat(5, {st_mode=S_IFREG|0644, st_size=32, ...}) = 0
unlink("/dev/shm/sem.WnwtPC")           = 0
close(5)                                = 0
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x769354965a10) = 10
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x769354965a10) = 11
read(0, abcdefg
"abcdefg\n", 1024)               = 8
futex(0x769354f11000, FUTEX_WAKE, 1) = 1
read(0, cdefgsb
"cdefgsb\n", 1024)               = 8
futex(0x769354f11000, FUTEX_WAKE, 1) = 1
read(0, asdawdawdsdajgjg
"asdawdawdsdajgjg\n", 1024)          = 17
futex(0x769354f11000, FUTEX_WAKE, 1) = 1
read(0, 123123123
"123123123\n", 1024)             = 10
futex(0x769354963000, FUTEX_WAKE, 1) = 1
read(0, STOP
"STOP\n", 1024)                  = 5
futex(0x769354f11000, FUTEX_WAKE, 1) = 1
futex(0x769354963000, FUTEX_WAKE, 1) = 1
wait4(10, NULL, 0, NULL)                = ? ERESTARTSYS (To be restarted if SA_RESTART is set)
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=11, si_uid=0, si_status=0,
si_utime=0, si_stime=0} ---
wait4(10, NULL, 0, NULL)                = 10
wait4(11, NULL, 0, NULL)                = 11
munmap(0x769354f13000, 4096)            = 0
munmap(0x769354f12000, 4096)            = 0
unlink("/dev/shm/shm1")                 = 0
unlink("/dev/shm/shm2")                 = 0
munmap(0x769354f11000, 32)              = 0
munmap(0x769354964000, 32)              = 0
munmap(0x769354963000, 32)              = 0
munmap(0x769354962000, 32)              = 0
unlink("/dev/shm/sem.sem_parent_ready_1") = 0
unlink("/dev/shm/sem.sem_child_ready_1") = 0
unlink("/dev/shm/sem.sem_parent_ready_2") = 0
unlink("/dev/shm/sem.sem_child_ready_2") = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```

Вывод

Я переделал первое задание с использованием shared memory в качестве IPC.