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: Савинов Н. О.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 28.12.24

Москва, 2024

Постановка задачи

Вариант 14.

Родительский процесс создает два дочерних процесса. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Child1 и Child2 можно «соединить» между собой дополнительным каналом. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child1 и child2 производят работу над строками. Child2 пересылает результат своей работы родительскому процессу. Родительский процесс полученный результат выводит в стандартный поток вывода. Child1 переводит строки в нижний регистр. Child2 убирает все задвоенные пробелы.

Общий метод и алгоритм решения

Использованные системные вызовы:

● pid_t fork(void); – создание дочернего процесса

● pid_t wait(int) - ожидание завершения дочерних процессов

● key_t ftok (const char *, int) – создание ключа System V IPC

● int shmget(key_t, size_t, int) – получение дескриптора (создание) разделяемого сегмента памяти

● void *shmat(int, const void*, int) – внесение разделяемого сегмента памяти в пространство имен процесса

● int shmdt(const void*) - удаленеи разделяемого сегмента памяти из пространства имен процесса

● int shmctl(int, int, struct shmid_ds *) - удаление разделяемого сегмента памяти

В рамках лабораторной работы я написал программу, которая использует механизмы межпроцессного взаимодействия через общую память в операционной системе Linux. Программа состоит из трех частей: родительского процесса и двух дочерних процессов. Задача заключалась в том, чтобы продемонстрировать взаимодействие между процессами, обработку строковых данных в общей памяти и использование системных вызовов для создания и управления общей памятью.

Родительский процесс создает файл для использования в качестве ключа для общедоступной памяти, с помощью ftok генерирует ключ для дальнейшего взаимодействия с общей памятью, создает сегмент общей памяти с использованием shmget и получает указатель на эту память через shmat. Так же читает строку с ввода пользователя, записывает её в общую память и затем запускает два дочерних процесса и ожидает завершения обоих дочерних процессов и выводит результат, который был изменен в общей памяти.

Дочерний процесс 1 (child1.c). Этот процесс читает данные из общей памяти, преобразует все символы строки в нижний регистр с помощью функции tolower и затем записывает измененную строку обратно в общую память.

Дочерний процесс 2 (child2.c). Этот процесс читает данные из общей памяти и убирает все задвоенные пробелы.

Код программы

parent.c

```
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <sys/shm.h>
#include <sys/ipc.h>
#include <sys/wait.h>
#include <errno.h>
#include <fcntl.h>

#define SHM_SIZE 1024

// Вывод ошибки и завершение программы
void handle_error(const char *msg) {
    const char *error_suffix = ": Ошибка\n";
    write(STDERR_FILENO, msg, strlen(msg));
    write(STDERR_FILENO, error_suffix, strlen(error_suffix));
    _exit(EXIT_FAILURE);
}

// Вывод сообщения в стандартный вывод
void write_message(const char *msg) {
    write(STDOUT_FILENO, msg, strlen(msg));
}

// Чтение строки из ввода
void read_message(char *buffer, size_t size) {
    ssize_t bytes_read = read(STDIN_FILENO, buffer, size - 1);
    if (bytes_read <= 0) handle_error("Ошибка чтения");
```

```
    buffer[bytes_read - 1] = '\0'; // Удаляем символ новой строки
}

int main() {
    // Создание файла для ключа
    int fd = open("shared_memory", O_CREAT | O_RDWR, 0666);
    if (fd == -1) handle_error("Ошибка создания файла");
    close(fd);

    // Создание ключа для разделяемой памяти
    key_t key = ftok("shared_memory", 65);
    if (key == -1) handle_error("Ошибка ftok");

    // Создание сегмента разделяемой памяти
    int shmid = shmget(key, SHM_SIZE, IPC_CREAT | 0666);
    if (shmid == -1) handle_error("Ошибка shmget");

    // Подключение к разделяемой памяти
    char *shared_memory = (char *)shmat(shmid, NULL, 0);
    if (shared_memory == (char *)-1) handle_error("Ошибка shmat");

    write_message("Введите строку (или пустую строку для выхода): ");
    char input_buffer[SHM_SIZE];

    // Основной цикл обработки ввода
    while (1) {
        read_message(input_buffer, SHM_SIZE);

        // Проверка на завершение
        if (strcmp(input_buffer, "") == 0) break;
```

```
// Копирование введенной строки в разделяемую память
strcpy(shared_memory, input_buffer);

// Запуск первого дочернего процесса
pid_t pid1 = fork();
if (pid1 == -1) handle_error("Ошибка fork (child1)");

if (pid1 == 0) {
    execl("./child1", "./child1", NULL);
    handle_error("Ошибка execl (child1)");
}

wait(NULL); // Ожидание завершения первого процесса

// Запуск второго дочернего процесса
pid_t pid2 = fork();
if (pid2 == -1) handle_error("Ошибка fork (child2)");

if (pid2 == 0) {
    execl("./child2", "./child2", NULL);
    handle_error("Ошибка execl (child2)");
}

wait(NULL); // Ожидание завершения второго процесса

// Вывод результата
write_message("Результат обработки: ");
write_message(shared_memory);
write_message("\nВведите строку (или пустую строку для выхода): ");
```

```
    }

    // Отключение от разделяемой памяти и удаление сегмента
    if (shmdt(shared_memory) == -1) handle_error("Ошибка shmdt");
    if (shmctl(shmid, IPC_RMID, NULL) == -1) handle_error("Ошибка shmctl");

    return 0;
}
```

child1.c

```
#include <unistd.h>
#include <stdlib.h>
#include <ctype.h>
#include <sys/shm.h>
#include <sys/ipc.h>
#include <errno.h>
#include <string.h>

#define SHM_SIZE 1024

void handle_error(const char *msg) {
    write(2, msg, strlen(msg));
    _exit(EXIT_FAILURE);
}

int main() {
    key_t key = ftok("shared_memory", 65);
    if (key == -1) handle_error("ftok");

    int shmid = shmget(key, SHM_SIZE, 0666);
    if (shmid == -1) handle_error("shmget");

    char *shared_memory = (char *)shmat(shmid, NULL, 0);
    if (shared_memory == (char *)-1) handle_error("shmat");

    for (int i = 0; shared_memory[i] != '\0'; i++)
        shared_memory[i] = tolower(shared_memory[i]);

    if (shmdt(shared_memory) == -1) handle_error("shmdt");
```

```
    return 0;
}
```

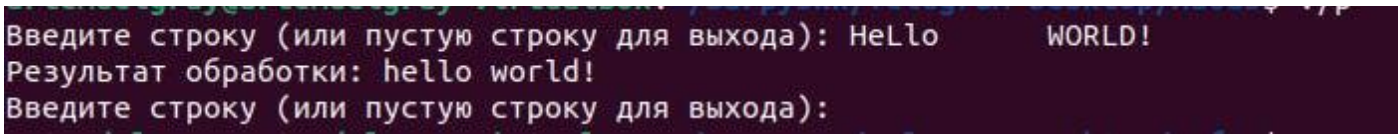
child2.c

```
#include <unistd.h>
#include <stdlib.h>
#include <sys/shm.h>
#include <sys/ipc.h>
#include <errno.h>
#include <string.h>

#define SHM_SIZE 1024

void handle_error(const char *msg) {
    write(2, msg, strlen(msg));
    _exit(EXIT_FAILURE);
}

int main() {
    // Получение ключа для разделяемой памяти
    key_t key = ftok("shared_memory", 65);
    if (key == -1) handle_error("Error: ftok failed\n");

    // Получение ID разделяемой памяти
    int shmid = shmget(key, SHM_SIZE, 0666);
    if (shmid == -1) handle_error("Error: shmget failed\n");

    // Присоединение к разделяемой памяти
    char *shared_memory = (char *)shmat(shmid, NULL, 0);
    if (shared_memory == (char *)-1) handle_error("Error: shmat failed\n");

    // Удаление подряд идущих пробелов
    char *read_ptr = shared_memory;
```

```
    char *write_ptr = shared_memory;
    int space_flag = 0;

    while (*read_ptr != '\0') {
        if (*read_ptr == ' ') {
            if (!space_flag) {
                *write_ptr++ = ' ';
                space_flag = 1; // Запоминаем, что пробел уже записан
            }
        } else {
            *write_ptr++ = *read_ptr;
            space_flag = 0; // Обнуляем флаг, так как встретился не пробел
        }
        read_ptr++;
    }
    *write_ptr = '\0'; // Завершаем строку

    // Отсоединение от разделяемой памяти
    if (shmdt(shared_memory) == -1) handle_error("Error: shmdt failed\n");

    return 0;
}
```

Протокол работы программы

Тестирование:



Strace:

```
strace -f ./p

execve("./p", ["./p"], 0x7ffe9c2d77f8 /* 46 vars */) = 0

brk(NULL)                               = 0x5b36a0c9a000

arch_prctl(0x3001 /* ARCH_??? */, 0x7fffe01e72a0) = -1 EINVAL (Недопустимый аргумент)

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x76af7bdda000

access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (Нет такого файла или каталога)

openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=58047, ...}, AT_EMPTY_PATH) = 0

mmap(NULL, 58047, PROT_READ, MAP_PRIVATE, 3, 0) = 0x76af7bdcb000

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\0P\237\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

pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48, 848) =
48

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0I\17\357\204\3$\f\221\2039x\324\224\323\236S"...,
68, 896) = 68

newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, AT_EMPTY_PATH) = 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, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x76af7ba00000

mprotect(0x76af7ba28000, 2023424, PROT_NONE) = 0

mmap(0x76af7ba28000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x28000) = 0x76af7ba28000

mmap(0x76af7bbbd000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) =
0x76af7bbbd000

mmap(0x76af7bc16000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x215000) = 0x76af7bc16000

mmap(0x76af7bc1c000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1,
0) = 0x76af7bc1c000

close(3)                                = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x76af7bdc8000

arch_prctl(ARCH_SET_FS, 0x76af7bdc8740) = 0

set_tid_address(0x76af7bdc8a10)         = 3583
```

```
set_robust_list(0x76af7bdc8a20, 24)     = 0

rseq(0x76af7bdc90e0, 0x20, 0, 0x53053053) = 0

mprotect(0x76af7bc16000, 16384, PROT_READ) = 0

mprotect(0x5b369fa97000, 4096, PROT_READ) = 0

mprotect(0x76af7be14000, 8192, PROT_READ) = 0

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

munmap(0x76af7bdcb000, 58047)           = 0

openat(AT_FDCWD, "shared_memory", O_RDWR|O_CREAT, 0666) = 3

close(3)                                = 0

newfstatat(AT_FDCWD, "shared_memory", {st_mode=S_IFREG|0664, st_size=0, ...}, 0) = 0

shmget(0x41033244, 1024, IPC_CREAT|0666) = 3

shmat(3, NULL, 0)                       = 0x76af7be13000

write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265
\321\201\321\202\321\200\320\276\320\272\321\203 (\320\270\320"..., 84Введите строку (или
пустую строку для выхода): ) = 84

read(0, Ro  fL

"Ro  fL\n", 1023)               = 7

clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLDstrace: Process
3584 attached

, child_tidptr=0x76af7bdc8a10) = 3584

[pid  3583] wait4(-1,  <unfinished ...>

[pid  3584] set_robust_list(0x76af7bdc8a20, 24) = 0

[pid  3584] execve("./child1", ["./child1"], 0x7fffe01e7478 /* 46 vars */) = 0

[pid  3584] brk(NULL)                   = 0x5e28ddaef000

[pid  3584] arch_prctl(0x3001 /* ARCH_??? */, 0x7fff23c70bb0) = -1 EINVAL (Недопустимый
аргумент)

[pid  3584] mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x752a69cbf000

[pid  3584] access("/etc/ld.so.preload", R_OK) = -1 ENOENT (Нет такого файла или каталога)

[pid  3584] openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

[pid  3584] newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=58047, ...}, AT_EMPTY_PATH) = 0

[pid  3584] mmap(NULL, 58047, PROT_READ, MAP_PRIVATE, 3, 0) = 0x752a69cb0000

[pid  3584] close(3)                    = 0

[pid  3584] openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
```

```
[pid  3584] read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\0\0\0\0\0"..., 832) = 832

[pid  3584] 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

[pid  3584] pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48, 848) = 48

[pid  3584] pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0I\17\357\204\3$\f\221\2039x\324\224\323\236S"..., 68, 896) = 68

[pid  3584] newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, AT_EMPTY_PATH) = 0

[pid  3584] 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

[pid  3584] mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x752a69a00000

[pid  3584] mprotect(0x752a69a28000, 2023424, PROT_NONE) = 0

[pid  3584] mmap(0x752a69a28000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x752a69a28000

[pid  3584] mmap(0x752a69bbd000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) = 0x752a69bbd000

[pid  3584] mmap(0x752a69c16000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x752a69c16000

[pid  3584] mmap(0x752a69c1c000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x752a69c1c000

[pid  3584] close(3)                    = 0

[pid  3584] mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x752a69cad000

[pid  3584] arch_prctl(ARCH_SET_FS, 0x752a69cad740) = 0

[pid  3584] set_tid_address(0x752a69cada10) = 3584

[pid  3584] set_robust_list(0x752a69cada20, 24) = 0

[pid  3584] rseq(0x752a69cae0e0, 0x20, 0, 0x53053053) = 0

[pid  3584] mprotect(0x752a69c16000, 16384, PROT_READ) = 0

[pid  3584] mprotect(0x5e28dbd9a000, 4096, PROT_READ) = 0

[pid  3584] mprotect(0x752a69cf9000, 8192, PROT_READ) = 0

[pid  3584] prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0

[pid  3584] munmap(0x752a69cb0000, 58047) = 0

[pid  3584] newfstatat(AT_FDCWD, "shared_memory", {st_mode=S_IFREG|0664, st_size=0, ...}, 0) = 0

[pid  3584] shmget(0x41033244, 1024, 0666) = 3
```

[pid 3584] shmat(3, NULL, 0)   = 0x752a69cf8000

[pid 3584] shmdt(0x752a69cf8000)   = 0

[pid 3584] exit_group(0)   = ?

[pid 3584] +++ exited with 0 +++

<... wait4 resumed>NULL, 0, NULL)   = 3584

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=3584, si_uid=1000, si_status=0, si_utime=0, si_stime=0} ---

clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLDstrace: Process 3585 attached

, child_tidptr=0x76af7bdc8a10) = 3585

[pid 3583] wait4(-1, <unfinished ...>

[pid 3585] set_robust_list(0x76af7bdc8a20, 24) = 0

[pid 3585] execve("./child2", ["./child2"], 0x7fffe01e7478 /* 46 vars */) = 0

[pid 3585] brk(NULL)   = 0x622e70b6d000

[pid 3585] arch_prctl(0x3001 /* ARCH_??? */, 0x7ffdf6b579e0) = -1 EINVAL (Недопустимый аргумент)

[pid 3585] mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ddc81137000

[pid 3585] access("/etc/ld.so.preload", R_OK) = -1 ENOENT (Нет такого файла или каталога)

[pid 3585] openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

[pid 3585] newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=58047, ...}, AT_EMPTY_PATH) = 0

[pid 3585] mmap(NULL, 58047, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7ddc81128000

[pid 3585] close(3)   = 0

[pid 3585] openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3

[pid 3585] read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\0\0\0\0\0"..., 832) = 832

[pid 3585] 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

[pid 3585] pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48, 848) = 48

[pid 3585] pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0I\17\357\204\3$\f\221\2039x\324\224\323\236S"..., 68, 896) = 68

[pid 3585] newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, AT_EMPTY_PATH) = 0

[pid 3585] 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

```
[pid  3585] mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7ddc80e00000

[pid  3585] mprotect(0x7ddc80e28000, 2023424, PROT_NONE) = 0

[pid  3585] mmap(0x7ddc80e28000, 1658880, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7ddc80e28000

[pid  3585] mmap(0x7ddc80fbd000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1bd000) = 0x7ddc80fbd000

[pid  3585] mmap(0x7ddc81016000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x7ddc81016000

[pid  3585] mmap(0x7ddc8101c000, 52816, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7ddc8101c000

[pid  3585] close(3)                    = 0

[pid  3585] mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7ddc81125000

[pid  3585] arch_prctl(ARCH_SET_FS, 0x7ddc81125740) = 0

[pid  3585] set_tid_address(0x7ddc81125a10) = 3585

[pid  3585] set_robust_list(0x7ddc81125a20, 24) = 0

[pid  3585] rseq(0x7ddc811260e0, 0x20, 0, 0x53053053) = 0

[pid  3585] mprotect(0x7ddc81016000, 16384, PROT_READ) = 0

[pid  3585] mprotect(0x622e6ed8e000, 4096, PROT_READ) = 0

[pid  3585] mprotect(0x7ddc81171000, 8192, PROT_READ) = 0

[pid  3585] prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY})
= 0

[pid  3585] munmap(0x7ddc81128000, 58047) = 0

[pid  3585] newfstatat(AT_FDCWD, "shared_memory", {st_mode=S_IFREG|0664, st_size=0, ...}, 0)
= 0

[pid  3585] shmget(0x41033244, 1024, 0666) = 3

[pid  3585] shmat(3, NULL, 0)           = 0x7ddc81170000

[pid  3585] shmdt(0x7ddc81170000)       = 0

[pid  3585] exit_group(0)               = ?

[pid  3585] +++ exited with 0 +++

<... wait4 resumed>NULL, 0, NULL)       = 3585

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=3585, si_uid=1000, si_status=0,
si_utime=0, si_stime=0} ---

write(1, "\320\240\320\265\320\267\321\203\320\273\321\214\321\202\320\260\321\202
\320\276\320\261\321\200\320\260\320\261\320\276\321"..., 39Результат обработки: ) = 39

write(1, "ro fl", 5ro fl)                   = 5
```

```
write(1, "\n\320\222\320\262\320\265\320\264\320\270\321\202\320\265
\321\201\321\202\321\200\320\276\320\272\321\203 (\320\270"..., 85

Введите строку (или пустую строку для выхода): ) = 85

read(0,

"\n", 1023)                     = 1

shmdt(0x76af7be13000)                   = 0

shmctl(3, IPC_RMID, NULL)               = 0

exit_group(0)                           = ?

+++ exited with 0 +++
```

Вывод

В ходе лабораторной работы была реализована программа, использующая общую память для межпроцессного взаимодействия. Родительский процесс записывает строку в общую память, а два дочерних процесса последовательно изменяют её, преобразуя символы в нижний регистр и заменяя пробелы на подчеркивания. Работа с системными вызовами, такими как shmget, shmat, и shmdt, а также синхронизация процессов через wait, позволили продемонстрировать основы взаимодействия между процессами. Лабораторная работа улучшила понимание механизмов работы с памятью и процессами в Linux.