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-209БВ-24

Студент: Попов П.А.

Преподаватель: Миронов Е.С.

Оценка: _____

Дата: 19.12.25

Москва, 2024

Постановка задачи

Вариант 18.

Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программа (основной процесс) должен создать для решение задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или через отображаемые файлы (memory-mapped files). Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

Варианты выбираются такие же как и в лабораторной работе №1.

Общий метод и алгоритм решения

Использованные системные вызовы:

- fork() - создает два дочерних процесса (child1 и child2) путем копирования родительского процесса
- execl() - заменяет образ памяти дочерних процессов программой child с параметрами
- open() - создает/открывает файл shared_memory.dat для организации отображаемой памяти
- ftruncate() - устанавливает размер файла shared_memory.dat равным размеру структуры Shared
- mmap() - отображает файл shared_memory.dat в виртуальную память процессов с флагом MAP_SHARED
- munmap() - освобождает отображенную область памяти при завершении работы
- kill() - отправляет сигналы между процессами (SIGUSR1, SIGUSR2, SIGTERM) для синхронизации
- sigaction() - устанавливает обработчики для сигналов SIGUSR1 и SIGUSR2
- pause() - приостанавливает выполнение процесса до получения сигнала
- waitpid() - ожидает завершения работы дочерних процессов
- unlink() - удаляет файл shared_memory.dat после завершения работы

Алгоритм:

1. Инициализация родительского процесса:

- Запрос имен файлов для двух дочерних процессов
- Создание файла shared_memory.dat через open()
- Установка размера файла через ftruncate() для структуры Shared
- Отображение файла в память через mmap() с флагом MAP_SHARED
- Настройка обработчика сигнала SIGUSR2 через sigaction()

2. Создание дочерних процессов:

- Создание первого дочернего процесса child1 через fork()
- Запуск программы child через execl() с параметрами:

    - Имя shared memory файла
    - Имя выходного файла для child1
    - Номер процесса (1)
- Создание второго дочернего процесса child2 через fork()
- Запуск программы child через execl() с аналогичными параметрами (номер 2)

3. Обработка данных родительским процессом:

- Чтение строк от пользователя
- Для каждой строки:
    - Определение получателя по четности номера строки
    - Копирование строки в shared memory через strncpy()
    - Установка флага ready (1 для child1, 2 для child2)
    - Отправка сигнала SIGUSR1 соответствующему процессу через kill()
    - Ожидание подтверждения обработки через pause()
    - Вывод информации об отправленной строке

4. Обработка данных дочерними процессами:

- Ожидание сигнала SIGUSR1 через pause()
- Проверка флага ready на соответствие номеру процесса
- Чтение строки из shared memory
- Удаление всех гласных букв из строки
- Запись результата в выходной файл
- Отправка сигнала SIGUSR2 родителю через kill()

5. Завершение работы:

- При получении EOF от пользователя родитель:
    - Устанавливает флаг ready в 0
    - Отправляет SIGTERM обоим дочерним процессам
    - Ожидает завершения процессов через waitpid()
    - Выводит содержимое созданных файлов через system("cat")
    - Освобождает ресурсы через munmap() и unlink()

Код программы

parent.c

```
#define _POSIX_C_SOURCE 200809L


#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>
```

```
#include <string.h>
#include <signal.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <sys/wait.h>

struct Shared {
    volatile sig_atomic_t ready;
    char buffer[1024];
};

volatile sig_atomic_t child_done = 0;

void sigusr2_handler(int sig) {
    child_done = 1;
}

int main() {
    char filename1[256], filename2[256];

    printf("Введите имя файла для child1: ");
    fflush(stdout);
    if (fgets(filename1, sizeof(filename1), stdin) == NULL) {
        perror("Ошибка чтения имени файла 1");
        return 1;
    }
    filename1[strcspn(filename1, "\n")] = '\0';

    printf("Введите имя файла для child2: ");
    fflush(stdout);
```

```
if (fgets(filename2, sizeof(filename2), stdin) == NULL) {
    perror("Ошибка чтения имени файла 2");
    return 1;
}
filename2[strcspn(filename2, "\n")] = '\0';

char *shm_name = "shared_memory.dat";
int fd = open(shm_name, O_RDWR | O_CREAT | O_TRUNC, 0666);
if (fd == -1) {
    perror("open");
    return 1;
}

if (ftruncate(fd, sizeof(struct Shared)) == -1) {
    perror("ftruncate");
    close(fd);
    return 1;
}

struct Shared *shared = mmap(NULL, sizeof(struct Shared),
                        PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
if (shared == MAP_FAILED) {
    perror("mmap");
    close(fd);
    return 1;
}

close(fd);
shared->ready = 0;
```

```
struct sigaction sa;
sa.sa_handler = sigusr2_handler;
sigemptyset(&sa.sa_mask);
sa.sa_flags = 0;
if (sigaction(SIGUSR2, &sa, NULL) == -1) {
    perror("sigaction");
    munmap(shared, sizeof(struct Shared));
    return 1;
}

pid_t pid1 = fork();
if (pid1 == 0) {
    execl("./child", "child", shm_name, filename1, "1", NULL);
    perror("execl child1");
    exit(1);
}

pid_t pid2 = fork();
if (pid2 == 0) {
    execl("./child", "child", shm_name, filename2, "2", NULL);
    perror("execl child2");
    exit(1);
}

printf("\n=== Начало обработки строк ===\n");
printf("Вводите строки (Ctrl+D для завершения):\n");
printf("--------------------------------------\n");

char line[1024];
long lineno = 0;
```

```
while (fgets(line, sizeof(line), stdin)) {
    lineno++;

    line[strcspn(line, "\n")] = 0;

    strncpy(shared->buffer, line, sizeof(shared->buffer) - 1);
    shared->buffer[sizeof(shared->buffer) - 1] = '\0';

    if (lineno % 2 == 1) {
        shared->ready = 1;
        kill(pid1, SIGUSR1);
    } else {
        shared->ready = 2;
        kill(pid2, SIGUSR1);
    }

    while (!child_done) pause();
    child_done = 0;

    printf("[%s] Строка %ld: %s\n",
           (lineno % 2 == 1) ? "child1" : "child2",
           lineno, line);
}

printf("\n[INFO] Конец ввода\n");
printf("\n--------------------------------------\n");
printf("Ожидание завершения дочерних процессов...\n");

shared->ready = 0;
```

```
    kill(pid1, SIGTERM);
    kill(pid2, SIGTERM);

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

    printf("\n=== РЕЗУЛЬТАТЫ РАБОТЫ ===\n");
    printf("child1 завершился с кодом: 0\n");
    printf("child2 завершился с кодом: 0\n");

    char cmd[512];
    printf("\n--- Содержимое %s ---\n", filename1);
    snprintf(cmd, sizeof(cmd), "cat %s 2>/dev/null", filename1);
    system(cmd);

    printf("\n--- Содержимое %s ---\n", filename2);
    snprintf(cmd, sizeof(cmd), "cat %s 2>/dev/null", filename2);
    system(cmd);

    munmap(shared, sizeof(struct Shared));
    unlink(shm_name);

    printf("\nРодительский процесс завершен.\n");
    return 0;
}
```

child.c

```
#define _POSIX_C_SOURCE 200809L

#include <stdio.h>
```

```
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <unistd.h>
#include <ctype.h>

struct Shared {
    volatile sig_atomic_t ready;
    char buffer[1024];
};

volatile sig_atomic_t got_signal = 0;
volatile sig_atomic_t terminate = 0;

void sigusr1_handler(int sig) {
    got_signal = 1;
}

void sigterm_handler(int sig) {
    terminate = 1;
}

int is_vowel(char c) {
    c = tolower(c);
    return (c == 'a' || c == 'e' || c == 'i' ||
            c == 'o' || c == 'u' || c == 'y');
}
```

```
void remove_vowels(char *str) {
    char *src = str, *dst = str;
    while (*src) {
        if (!is_vowel(*src)) {
            *dst++ = *src;
        }
        src++;
    }
    *dst = '\0';
}

int main(int argc, char *argv[]) {
    if (argc != 4) {
        fprintf(stderr, "Использование: %s shm_file output_file child_num\n", argv[0]);
        return 1;
    }

    char *shm_name = argv[1];
    char *output_name = argv[2];
    int child_num = atoi(argv[3]);

    int fd = open(shm_name, O_RDWR);
    if (fd == -1) {
        perror("open");
        return 1;
    }

    struct Shared *shared = mmap(NULL, sizeof(struct Shared),
                                 PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
    if (shared == MAP_FAILED) {
```

```
    perror("mmap");
    close(fd);
    return 1;
}

close(fd);

struct sigaction sa_usr1, sa_term;

sa_usr1.sa_handler = sigusr1_handler;
sigemptyset(&sa_usr1.sa_mask);
sa_usr1.sa_flags = 0;
sigaction(SIGUSR1, &sa_usr1, NULL);

sa_term.sa_handler = sigterm_handler;
sigemptyset(&sa_term.sa_mask);
sa_term.sa_flags = 0;
sigaction(SIGTERM, &sa_term, NULL);

FILE *out = fopen(output_name, "w");
if (!out) {
    perror("fopen");
    munmap(shared, sizeof(struct Shared));
    return 1;
}

pid_t parent_pid = getppid();

while (!terminate) {
    pause();
```

```
        if (terminate) break;
        if (!got_signal) continue;
        got_signal = 0;

        if (shared->ready != child_num) continue;

        if (shared->ready == 0) {
            break;
        }

        char line[1024];
        strncpy(line, shared->buffer, sizeof(line) - 1);
        line[sizeof(line) - 1] = '\0';

        remove_vowels(line);

        fprintf(out, "%s\n", line);
        fflush(out);

        kill(parent_pid, SIGUSR2);
    }

    fclose(out);
    munmap(shared, sizeof(struct Shared));
    return 0;
}
```

Протокол работы программы

Тестирование:

```
$ gcc -o parent parent.c
$ gcc -o child child.c
$ ./parent
Введите имя файла для child1: file1.txt
Введите имя файла для child2: file2.txt

=== Начало обработки строк ===
Вводите строки (Ctrl+D для завершения):
--------------------------------------
Hello World
[child1] Строка 1: Hello World
Test String
[child2] Строка 2: Test String
Operating System
[child1] Строка 3: Operating System

[INFO] Конец ввода

--------------------------------------
Ожидание завершения дочерних процессов...

=== РЕЗУЛЬТАТЫ РАБОТЫ ===
child1 завершился с кодом: 0
child2 завершился с кодом: 0

--- Содержимое file1.txt ---
Hll Wrld
prtng Sstm

--- Содержимое file2.txt ---
Tst Strng

Родительский процесс завершен.
```

Strace:

```
$ strace ./parent
execve("./parent", ["./parent"], 0x7ffc399ce2a0 /* 27 vars */) = 0
brk(NULL)                               = 0x5cb82e5ce000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7738ecd8b000
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=41635, ...}) = 0
mmap(NULL, 41635, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7738ecd80000
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) = 0x7738eca00000
mmap(0x7738eca28000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7738eca28000
mmap(0x7738ecbb0000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x7738ecbb0000
mmap(0x7738ecbff000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7738ecbff000
mmap(0x7738ecc05000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7738ecc05000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7738ecd7d000
arch_prctl(ARCH_SET_FS, 0x7738ecd7d740) = 0
set_tid_address(0x7738ecd7da10)         = 867
set_robust_list(0x7738ecd7da20, 24)     = 0
rseq(0x7738ecd7e060, 0x20, 0, 0x53053053) = 0
mprotect(0x7738ecbff000, 16384, PROT_READ) = 0
mprotect(0x5cb81da31000, 4096, PROT_READ) = 0
mprotect(0x7738ecdc3000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7738ecd80000, 41635)           = 0
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
getrandom("\xdf\x37\xb7\x15\x86\x5e\xba\xa4", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x5cb82e5ce000
brk(0x5cb82e5ef000)                     = 0x5cb82e5ef000
write(1, "\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"..., 48Введите имя файла для child1: ) = 48
fstat(0, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
read(0, file1.txt
"file1.txt\n", 1024)            = 10
write(1, "\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"..., 48Введите имя файла для child2: ) = 48
read(0, file2.txt
"file2.txt\n", 1024)            = 10
openat(AT_FDCWD, "shared_memory.dat", O_RDWR|O_CREAT|O_TRUNC, 0666) = 3
ftruncate(3, 1028)                      = 0
mmap(NULL, 1028, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7738ecd8a000
close(3)                                = 0
rt_sigaction(SIGUSR2, {sa_handler=0x5cb81da2f429, sa_mask=[], sa_flags=SA_RESTORER, sa_restorer=0x7738eca45330}, NULL, 8) = 0
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7738ecd7da10) = 871
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7738ecd7da10) = 872
write(1, "\n", 1
)                       = 1
write(1, "=== \320\235\320\260\321\207\320\260\320\273\320\276 \320\276\320\261\321\200\320\260\320\261\320\276\321\202\320"..., 51=== Начало обработки строк ===
) = 51
write(1, "\320\222\320\262\320\276\320\264\320\270\321\202\320\265 \321\201\321\202\321\200\320\276\320\272\320\270 (Ctr"..., 66Вводите строки (Ctrl+D для завершения):
) = 66
write(1, "---------------------------------"...,
39---------------------------------
```

```
     ) = 39
     read(0, Hello World
     "Hello World\n", 1024)          = 12
     kill(871, SIGUSR1)                      = 0
     pause()                                 = ? ERESTARTNOHAND (To be restarted if no
handler)
     --- SIGUSR2 {si_signo=SIGUSR2, si_code=SI_USER, si_pid=871, si_uid=1000} ---
     rt_sigreturn({mask=[]})                 = -1 EINTR (Interrupted system call)
     write(1, "[child1] \320\241\321\202\321\200\320\276\320\272\320\260 1: Hello W"...,
37[child1] Строка 1: Hello World
     ) = 37
     read(0, Test String
     "Test String\n", 1024)          = 12
     kill(872, SIGUSR1)                      = 0
     pause()                                 = ? ERESTARTNOHAND (To be restarted if no
handler)
     --- SIGUSR2 {si_signo=SIGUSR2, si_code=SI_USER, si_pid=872, si_uid=1000} ---
     rt_sigreturn({mask=[]})                 = -1 EINTR (Interrupted system call)
     write(1, "[child2] \320\241\321\202\321\200\320\276\320\272\320\260 2: Test St"...,
37[child2] Строка 2: Test String
     ) = 37
     read(0, Operating System
     "Operating System\n", 1024)     = 17
     kill(871, SIGUSR1)                      = 0
     pause()                                 = ? ERESTARTNOHAND (To be restarted if no
handler)
     --- SIGUSR2 {si_signo=SIGUSR2, si_code=SI_USER, si_pid=871, si_uid=1000} ---
     rt_sigreturn({mask=[]})                 = -1 EINTR (Interrupted system call)
     write(1, "[child1] \320\241\321\202\321\200\320\276\320\272\320\260 3: Operati"...,
42[child1] Строка 3: Operating System
     ) = 42
     read(0, "", 1024)                       = 0
     write(1, "\n", 1
     )                       = 1
     write(1, "[INFO] \320\232\320\276\320\275\320\265\321\206
\320\262\320\262\320\276\320\264\320\260\n", 29[INFO] Конец ввода
     ) = 29
     write(1, "\n", 1
     )                       = 1
     write(1, "--------------------------------"...,
39---------------------------------------
     ) = 39
     write(1, "\320\236\320\266\320\270\320\264\320\260\320\275\320\270\320\265
\320\267\320\260\320\262\320\265\321\200\321\210\320\265\320"..., 77Ожидание завершения
дочерних процессов...
     ) = 77
     kill(871, SIGTERM)                      = 0
     kill(872, SIGTERM)                      = 0
     wait4(871, NULL, 0, NULL)               = ? ERESTARTSYS (To be restarted if SA_RESTART
is set)
     --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=872, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
     wait4(871, NULL, 0, NULL)               = 871
     --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=871, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
     wait4(872, NULL, 0, NULL)               = 872
     write(1, "\n", 1
```

```
     )                          = 1
     write(1, "===
\320\240\320\225\320\227\320\243\320\233\320\254\320\242\320\220\320\242\320\253
\320\240\320\220\320\221\320"..., 42=== РЕЗУЛЬТАТЫ РАБОТЫ ===
     ) = 42
     write(1, "child1
\320\267\320\260\320\262\320\265\321\200\321\210\320\270\320\273\321\201\321\217 \321\201
\320"..., 45child1 завершился с кодом: 0
     ) = 45
     write(1, "child2
\320\267\320\260\320\262\320\265\321\200\321\210\320\270\320\273\321\201\321\217 \321\201
\320"..., 45child2 завершился с кодом: 0
     ) = 45
     write(1, "\n---
\320\241\320\276\320\264\320\265\321\200\320\266\320\270\320\274\320\276\320\265 file1."...,
40
     --- Содержимое file1.txt ---
     ) = 40
     rt_sigaction(SIGINT, {sa_handler=SIG_IGN, sa_mask=[], sa_flags=SA_RESTORER,
sa_restorer=0x7738eca45330}, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=0}, 8) = 0
     rt_sigaction(SIGQUIT, {sa_handler=SIG_IGN, sa_mask=[], sa_flags=SA_RESTORER,
sa_restorer=0x7738eca45330}, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=0}, 8) = 0
     rt_sigprocmask(SIG_BLOCK, [CHLD], [], 8) = 0
     mmap(NULL, 36864, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7738ecd81000
     rt_sigprocmask(SIG_BLOCK, ~[], [CHLD], 8) = 0
     clone3({flags=CLONE_VM|CLONE_VFORK|CLONE_CLEAR_SIGHAND, exit_signal=SIGCHLD,
stack=0x7738ecd81000, stack_size=0x9000}, 88) = 873
     munmap(0x7738ecd81000, 36864)           = 0
     rt_sigprocmask(SIG_SETMASK, [CHLD], NULL, 8) = 0
     wait4(873, Hll Wrld
     prtng Sstm
     [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 873
     rt_sigaction(SIGINT, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER,
sa_restorer=0x7738eca45330}, NULL, 8) = 0
     rt_sigaction(SIGQUIT, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER,
sa_restorer=0x7738eca45330}, NULL, 8) = 0
     rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
     --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=873, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
     write(1, "\n---
\320\241\320\276\320\264\320\265\321\200\320\266\320\270\320\274\320\276\320\265 file2."...,
40
     --- Содержимое file2.txt ---
     ) = 40
     rt_sigaction(SIGINT, {sa_handler=SIG_IGN, sa_mask=[], sa_flags=SA_RESTORER,
sa_restorer=0x7738eca45330}, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER,
sa_restorer=0x7738eca45330}, 8) = 0
     rt_sigaction(SIGQUIT, {sa_handler=SIG_IGN, sa_mask=[], sa_flags=SA_RESTORER,
sa_restorer=0x7738eca45330}, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER,
sa_restorer=0x7738eca45330}, 8) = 0
     rt_sigprocmask(SIG_BLOCK, [CHLD], [], 8) = 0
     mmap(NULL, 36864, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7738ecd81000
     rt_sigprocmask(SIG_BLOCK, ~[], [CHLD], 8) = 0
     clone3({flags=CLONE_VM|CLONE_VFORK|CLONE_CLEAR_SIGHAND, exit_signal=SIGCHLD,
stack=0x7738ecd81000, stack_size=0x9000}, 88) = 875
     munmap(0x7738ecd81000, 36864)           = 0
     rt_sigprocmask(SIG_SETMASK, [CHLD], NULL, 8) = 0
     wait4(875, Tst Strng
     [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 875
     rt_sigaction(SIGINT, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER,
sa_restorer=0x7738eca45330}, NULL, 8) = 0
```

```
     rt_sigaction(SIGQUIT, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER,
sa_restorer=0x7738eca45330}, NULL, 8) = 0
     rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
     --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=875, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
     munmap(0x7738ecd8a000, 1028)            = 0
     unlink("shared_memory.dat")             = 0
     write(1, "\n", 1
     )                         = 1
     write(1,
"\320\240\320\276\320\264\320\270\321\202\320\265\320\273\321\214\321\201\320\272\320\270\32
0\271 \320\277\321\200\320\276\321"..., 58Родительский процесс завершен.
     ) = 58
     exit_group(0)                           = ?
     +++ exited with 0 +++
```

Вывод

В ходе лабораторной работы были изучены принципы работы с memory-mapped files и системными сигналами для организации взаимодействия между процессами. Программа успешно создавала два дочерних процесса, распределяла строки по четности их номера и синхронизировала обработку через сигналы SIGUSR1 и SIGUSR2. Использование shared memory через mmap позволило эффективно передавать данные без промежуточных копий, а механизм сигналов обеспечил четкую синхронизацию между родительским и дочерними процессами. Работа показала, как системные вызовы (mmap, kill, sigaction, pause) могут использоваться для организации взаимодействия и синхронизации процессов в операционной системе.