Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №1 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 каналы (pipe). Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

Группа вариантов 5: родительский процесс создает два дочерних процесса. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись для child1. Аналогично для второй строки и процесса child2. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1 или в pipe2 в зависимости от правила фильтрации. Процесс child1 и child2 производят работу над строками. Процессы пишут результаты своей работы в стандартный вывод.

Вариант 18: правило фильтрации: нечетные строки отправляются в pipe1, четные в pipe2. Дочерние процессы удаляют все гласные из строк.

Общий метод и алгоритм решения

Использованные системные вызовы:

- `fork()` — создает новый процесс; используется дважды для создания двух дочерних процессов child1 и child2
- `pipe()` — создает канал для межпроцессного взаимодействия; используется дважды для создания pipe1 (child1) и pipe2 (child2)
- `dup2()` — переназначает файловые дескрипторы; дочерние процессы перенаправляют стандартный ввод на свои каналы
- `execl()` — заменяет код текущего процесса новой программой child, которая удаляет гласные из строк
- `waitpid()` — приостанавливает родительский процесс до завершения дочерних
- `write()` — записывает данные в канал; родитель отправляет строки дочерним процессам
- `getline()` — читает строки произвольной длины от пользователя и из каналов
- `open()` — открывает выходные файлы для записи обработанных строк
- `close()` — закрывает неиспользуемые концы каналов, сигнализируя о завершении передачи
- `exit()` — завершает процессы с кодом возврата

Алгоритм:

Родитель создает два канала через `pipe()` и два дочерних процесса через `fork()`

1. Каждый дочерний процесс перенаправляет свой стандартный ввод на соответствующий канал с помощью dup2() и запускает программу child через execl()
2. Родитель читает строки от пользователя с помощью getline() и отправляет нечетные строки в pipe1, четные — в pipe2 через write()
3. Дочерние процессы читают строки из стандартного ввода (перенаправленного канала), удаляют гласные буквы и записывают результат в свои файлы
4. После завершения ввода (EOF) родитель закрывает каналы с помощью close() и ожидает завершения дочерних процессов через waitpid()
5. Родитель выводит статус завершения и содержимое результирующих файлов, затем завершает работу

Код программы

parent.c

```
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

#include <string.h>

#include <errno.h>


static ssize_t write_all(int fd, const void *buf, size_t count) {
    const char *p = buf;
    size_t left = count;
    while (left > 0) {
        ssize_t w = write(fd, p, left);
        if (w < 0) {
            if (errno == EINTR) continue;
            return -1;
        }
        left -= (size_t)w;
        p += w;
    }
    return (ssize_t)count;
```

```
}

int main() {
    char *filename1 = NULL, *filename2 = NULL;
    size_t fncap = 0;
    ssize_t r;

    printf("Введите имя файла для child1: ");
    fflush(stdout);
    r = getline(&filename1, &fncap, stdin);
    if (r <= 0) {
        perror("Ошибка чтения имени файла 1");
        free(filename1);
        return 1;
    }
    if (filename1[r-1] == '\n') filename1[r-1] = '\0';

    printf("Введите имя файла для child2: ");
    fflush(stdout);
    fncap = 0;
    r = getline(&filename2, &fncap, stdin);
    if (r <= 0) {
        perror("Ошибка чтения имени файла 2");
        free(filename1);
        free(filename2);
        return 1;
    }
    if (filename2[r-1] == '\n') filename2[r-1] = '\0';

    int pipe1[2], pipe2[2];
```

```
if (pipe(pipe1) == -1) {
    perror("Ошибка создания pipe1");
    free(filename1);
    free(filename2);
    return 1;
}
if (pipe(pipe2) == -1) {
    perror("Ошибка создания pipe2");
    close(pipe1[0]);
    close(pipe1[1]);
    free(filename1);
    free(filename2);
    return 1;
}

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

pid_t pid1 = fork();
if (pid1 < 0) {
    perror("Ошибка fork для child1");
    close(pipe1[0]); close(pipe1[1]); close(pipe2[0]); close(pipe2[1]);
    free(filename1); free(filename2);
    return 1;
}

if (pid1 == 0) {
    if (dup2(pipe1[0], STDIN_FILENO) == -1) {
        perror("child1: ошибка dup2");
```

```
        _exit(1);
    }
    close(pipe1[0]); close(pipe1[1]);
    close(pipe2[0]); close(pipe2[1]);

    execl("./child", "child", filename1, (char *)NULL);
    perror("child1: ошибка execl");
    _exit(1);
}

close(pipe1[0]);

pid_t pid2 = fork();
if (pid2 < 0) {
    perror("Ошибка fork для child2");
    close(pipe1[1]); close(pipe2[0]); close(pipe2[1]);
    free(filename1); free(filename2);
    return 1;
}

if (pid2 == 0) {
    if (dup2(pipe2[0], STDIN_FILENO) == -1) {
        perror("child2: ошибка dup2");
        _exit(1);
    }
    close(pipe2[0]); close(pipe2[1]);
    close(pipe1[0]); close(pipe1[1]);

    execl("./child", "child", filename2, (char *)NULL);
    perror("child2: ошибка execl");
```

```
    _exit(1);
}

close(pipe2[0]);

char *line = NULL;
size_t linecap = 0;
long lineno = 0;

while (getline(&line, &linecap, stdin) != -1) {
    ++lineno;

    int target_fd = (lineno % 2 == 1) ? pipe1[1] : pipe2[1];
    const char *child_name = (lineno % 2 == 1) ? "child1" : "child2";

    if (write_all(target_fd, line, strlen(line)) == -1) {
        fprintf(stderr, "Ошибка записи в %s: %s\n", child_name, strerror(errno));
    }

    size_t len = strlen(line);
    if (len > 0 && line[len-1] == '\n') line[len-1] = '\0';
    printf("[%s] Строка %ld: %s\n", child_name, lineno, line);
}

free(line);

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

```
    close(pipe1[1]);
    close(pipe2[1]);

    int status1, status2;
    if (waitpid(pid1, &status1, 0) == -1) perror("Ошибка waitpid для child1");
    if (waitpid(pid2, &status2, 0) == -1) perror("Ошибка waitpid для child2");

    printf("\n=== РЕЗУЛЬТАТЫ РАБОТЫ ===\n");
    printf("child1 завершился с кодом: %d\n", WEXITSTATUS(status1));
    printf("child2 завершился с кодом: %d\n", WEXITSTATUS(status2));
    printf("\nРезультаты записаны в файлы:\n");
    printf("• %s (child1, нечетные строки)\n", filename1);
    printf("• %s (child2, четные строки)\n", filename2);

    printf("\n--- Содержимое %s ---\n", filename1);
    char cmd[1024];
    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);

    free(filename1);
    free(filename2);
    printf("\nРодительский процесс завершен.\n");
    return 0;
}
```

child.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

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;
    char *dst = str;

    while (*src) {
        if (!is_vowel(*src)) {
            *dst = *src;
            dst++;
        }
        src++;
    }
    *dst = '\0';
}

int main(int argc, char *argv[]) {
    if (argc < 2) {
        fprintf(stderr, "Использование: %s output_filename\n", argv[0]);
        return 1;
    }
```

```
const char *outname = argv[1];
FILE *out = fopen(outname, "w");
if (!out) {
    perror("Ошибка открытия файла");
    return 1;
}

char *line = NULL;
size_t cap = 0;

while (getline(&line, &cap, stdin) != -1) {
    size_t len = strlen(line);
    if (len > 0 && line[len-1] == '\n') {
        line[len-1] = '\0';
        len--;
    }

    char original[1024];
    if (len >= sizeof(original)) len = sizeof(original) - 1;
    strncpy(original, line, len);
    original[len] = '\0';

    remove_vowels(line);

    printf("%s\n", line);
    fflush(stdout);

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

```
    }

    free(line);
    fclose(out);
    return 0;
}
```

Протокол работы программы

Тестирование:

```
$ make
gcc -Wall -Wextra -std=c99 -D_POSIX_C_SOURCE=200809L parent.c -o parent
gcc -Wall -Wextra -std=c99 -D_POSIX_C_SOURCE=200809L child.c -o child
$ ./parent
Введите имя файла для child1: file01.txt
Введите имя файла для child2: file02.txt

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

[INFO] Конец ввода (EOF)

------------------------------------------
```

Ожидание завершения дочерних процессов...

=== РЕЗУЛЬТАТЫ РАБОТЫ ===

child1 завершился с кодом: 0

child2 завершился с кодом: 0

Результаты записаны в файлы:

• file01.txt (child1, нечетные строки)

• file02.txt (child2, четные строки)

--- Содержимое file01.txt ---

Hll Wrld

Tst Strng

--- Содержимое file02.txt ---

prtng Sstm

Родительский процесс завершен.

Strace:

```
execve("./parent", ["./parent", "-f"], 0x7ffecc741c28 /* 28 vars */) = 0
brk(NULL)                               = 0x6301051f9000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
0) = 0x735feeee1000
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) = 0x735feeed6000
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) =
0x735feec00000
mmap(0x735feec28000, 1605632, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x735feec28000
mmap(0x735feedb0000, 323584, PROT_READ,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x735feedb0000
```

```
mmap(0x735feedff000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x735feedff000
mmap(0x735feee05000, 52624, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x735feee05000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
0) = 0x735feeed3000
arch_prctl(ARCH_SET_FS, 0x735feeed3740) = 0
set_tid_address(0x735feeed3a10)         = 1445
set_robust_list(0x735feeed3a20, 24)     = 0
rseq(0x735feeed4060, 0x20, 0, 0x53053053) = 0
mprotect(0x735feedff000, 16384, PROT_READ) = 0
mprotect(0x6300d497b000, 4096, PROT_READ) = 0
mprotect(0x735feef19000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) =
0
munmap(0x735feeed6000, 41635)           = 0
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
getrandom("\x71\x0d\xa4\xb8\x15\x74\x8e\x3c", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x6301051f9000
brk(0x63010521a000)                     = 0x63010521a000
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
pipe2([3, 4], 0)                        = 0
pipe2([5, 6], 0)                        = 0
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"..., 77Вводите строки (Ctrl+D для
завершения ввода):
) = 77
write(1, "--------------------------------"..., 45----------------------------------------
) = 45
clone(child_stack=NULL,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x735feeed3a10) = 1448
close(3)                                = 0
clone(child_stack=NULL,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x735feeed3a10) = 1449
close(5)                                = 0
read(0, Hello World
"Hello World\n", 1024)          = 12
```

```
write(4, "Hello World\n", 12)           = 12
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
write(6, "Test String\n", 12)           = 12
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
write(4, "Operating System\n", 17)      = 17
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 (EO"..., 35[INFO] Конец ввода (EOF)
) = 35
write(1, "\n", 1
)                       = 1
write(1, "--------------------------------"..., 45---------------------------------------------
) = 45
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
close(4)                                = 0
close(6)                                = 0
wait4(1448, 0x7ffd0c291b64, 0, NULL)    = ? ERESTARTSYS (To be restarted if SA_RESTART is set)
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=1449, si_uid=1000, si_status=0, si_utime=0, si_stime=0} ---
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=1448, si_uid=1000, si_status=0, si_utime=0, si_stime=0} ---
wait4(1448, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 1448
wait4(1449, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 1449
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", 1
```

```
        )                       = 1
        write(1, "\320\240\320\265\320\267\321\203\320\273\321\214\321\202\320\260\321\202\321\213
\320\267\320\260\320\277\320\270\321\201\320"..., 53Результаты записаны в файлы:
        ) = 53
        write(1, "\342\200\242 file1.txt (child1, \320\275\320\265\321\207\320\265\321"..., 54• file1.txt
(child1, нечетные строки)
        ) = 54
        write(1, "\342\200\242 file2.txt (child2, \321\207\320\265\321\202\320\275\321"..., 50• file2.txt
(child2, четные строки)
        ) = 50
        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=0x735feec45330}, {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=0x735feec45330}, {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) = 0x735feeed8000
        rt_sigprocmask(SIG_BLOCK, ~[], [CHLD], 8) = 0
        clone3({flags=CLONE_VM|CLONE_VFORK|CLONE_CLEAR_SIGHAND,
exit_signal=SIGCHLD, stack=0x735feeed8000, stack_size=0x9000}, 88) = 1450
        munmap(0x735feeed8000, 36864)           = 0
        rt_sigprocmask(SIG_SETMASK, [CHLD], NULL, 8) = 0
        wait4(1450, Hll Wrld
        prtng Sstm
        [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 1450
        rt_sigaction(SIGINT, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER,
sa_restorer=0x735feec45330}, NULL, 8) = 0
        rt_sigaction(SIGQUIT, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER,
sa_restorer=0x735feec45330}, NULL, 8) = 0
        rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
        --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=1450, 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=0x735feec45330}, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER,
sa_restorer=0x735feec45330}, 8) = 0
        rt_sigaction(SIGQUIT, {sa_handler=SIG_IGN, sa_mask=[], sa_flags=SA_RESTORER,
sa_restorer=0x735feec45330}, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER,
sa_restorer=0x735feec45330}, 8) = 0
        rt_sigprocmask(SIG_BLOCK, [CHLD], [], 8) = 0
        mmap(NULL, 36864, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x735feeed8000
        rt_sigprocmask(SIG_BLOCK, ~[], [CHLD], 8) = 0
        clone3({flags=CLONE_VM|CLONE_VFORK|CLONE_CLEAR_SIGHAND,
exit_signal=SIGCHLD, stack=0x735feeed8000, stack_size=0x9000}, 88) = 1452
        munmap(0x735feeed8000, 36864)           = 0
        rt_sigprocmask(SIG_SETMASK, [CHLD], NULL, 8) = 0
        wait4(1452, Tst Strng
        [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 1452
```

```
        rt_sigaction(SIGINT, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER,
sa_restorer=0x735feec45330}, NULL, 8) = 0
        rt_sigaction(SIGQUIT, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER,
sa_restorer=0x735feec45330}, NULL, 8) = 0
        rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
        --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=1452, si_uid=1000,
si_status=0, si_utime=0, si_stime=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\320\2
71 \320\277\321\200\320\276\321"..., 58Родительский процесс завершен.
        ) = 58
        exit_group(0)                   = ?
        +++ exited with 0 +++
```

Вывод

Лабораторная работа продемонстрировала использование системных вызовов для создания процессов и обмена данными через каналы. Родительский процесс создает двух дочерних, распределяет строки по четности, а дочерние процессы удаляют гласные буквы и записывают результаты в файлы. Программа демонстрирует корректное межпроцессное взаимодействие и обработку системных ошибок.