Федеральное государственное автономное образовательное учреждение высшего образования
"Национальный исследовательский университет "Высшая школа экономики"
Московский институт электроники и математики
им. А. Н. Тихонова НИУ ВШЭ
Департамент компьютерной инженерии

Лабораторная работа № 5
по теме:
«Организация взаимодействия процессов через pipe и FIFO»

По дисциплине

«Операционные системы»

Группа: **БПМ 243**

Студент: **Николаев Степан Денисович**

Москва 2026

Ход работы

Создал pipe_example.c. Программа создаёт pipe и два дочерних процесса: один записывает сообщение, другой читает. Родитель закрывает канал и ждёт их завершения. Скомпилировал программу и запустил два раза.

```
marmos@marmos-QEMU-Virtual-Machine:~/practice5$ nano pipe_example.c
marmos@marmos-QEMU-Virtual-Machine:~/practice5$ gcc pipe_example.c -o pipe_example
marmos@marmos-QEMU-Virtual-Machine:~/practice5$ ./pipe_example
Отправлено: Привет через pipe!
Получено: Привет через pipe!
marmos@marmos-QEMU-Virtual-Machine:~/practice5$ ./pipe_example
Отправлено: Привет через pipe!
Получено: Привет через pipe!
marmos@marmos-QEMU-Virtual-Machine:~/practice5$
```

Рис. 1. Компиляция и проверка pipe.

В fifo_example.c создал именованный канал myfifo. Один дочерний процесс пишет в него, второй читает. После завершения обмена родитель удаляет FIFO. В обоих запусках сообщение передалось; порядок строк «Отправлено» и «Получено» различался.

```
marmos@marmos-QEMU-Virtual-Machine:~/practice5$ nano fifo_example.c
marmos@marmos-QEMU-Virtual-Machine:~/practice5$ gcc fifo_example.c -o fifo_example
marmos@marmos-QEMU-Virtual-Machine:~/practice5$ ./fifo_example
Отправлено: Привет через FIFO!
Получено: Привет через FIFO!
marmos@marmos-QEMU-Virtual-Machine:~/practice5$ ./fifo_example
Получено: Привет через FIFO!
Отправлено: Привет через FIFO!
marmos@marmos-QEMU-Virtual-Machine:~/practice5$
```

Рис. 2. Компиляция и проверка FIFO.

Через wc -c проверил размер сообщений: по 29 байт текста. В коде передаётся sizeof(message), то есть 30 байт вместе с завершающим \0. Расчётный объём за 1000 запусков — 30 000 байт для каждого варианта.

```
marmos@marmos-QEMU-Virtual-Machine:~/practice5$ printf "Привет через pipe!" | wc -c
29
marmos@marmos-QEMU-Virtual-Machine:~/practice5$ printf "Привет через FIFO!" | wc -c
29
```

Рис. 3. Проверка размера сообщений.

Сравнение результатов

Запустил каждую программу 1000 раз через /usr/bin/time -v, вывод направил в /dev/null. Замер включает запуск процессов, создание каналов и завершение программ, а не только передачу сообщения.

```
marmos@marmos-QEMU-Virtual-Machine:~/practice5$ /usr/bin/time -v sh -c 'for i in $(seq 1 1000); do
./pipe_example > /dev/null; done'
        Command being timed: "sh -c for i in $(seq 1 1000); do ./pipe_example > /dev/null; done"
        User time (seconds): 0.51
        System time (seconds): 0.44
        Percent of CPU this job got: 118%
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.81
        Average shared text size (kbytes): 0
        Average unshared data size (kbytes): 0
        Average stack size (kbytes): 0
        Average total size (kbytes): 0
        Maximum resident set size (kbytes): 6688
```

Рис. 4. Замер 1000 запусков pipe (фрагмент вывода).

```
marmos@marmos-QEMU-Virtual-Machine:~/practice5$ /usr/bin/time -v sh -c 'for i in $(seq 1 1000); do
./fifo_example > /dev/null; done'
        Command being timed: "sh -c for i in $(seq 1 1000); do ./fifo_example > /dev/null; done"
        User time (seconds): 0.62
        System time (seconds): 0.63
        Percent of CPU this job got: 96%
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:01.30
        Average shared text size (kbytes): 0
        Average unshared data size (kbytes): 0
        Average stack size (kbytes): 0
        Average total size (kbytes): 0
        Maximum resident set size (kbytes): 6824
```

Рис. 5. Замер 1000 запусков FIFO (фрагмент вывода).

Показатель	**Pipe**	**FIFO**
Время 1000 запусков, с	0,81	1,30
CPU по выводу time	118 %	96 %
Макс. RSS по выводу time, Кбайт	6688	6824
Данные за один запуск, байт	30	30
Данные за 1000 запусков, байт	30 000	30 000

CPU выше 100 % возможен при одновременной работе нескольких процессов на разных ядрах.

Вывод. В этом тесте pipe оказался быстрее: 0,81 с против 1,30 с. Показатели памяти близкие. Pipe удобен для обмена между родственными процессами и не требует имени в файловой системе. FIFO подходит и для независимых процессов, но требует создания и удаления именованного канала. Результат относится к этим программам и объёму данных, а не к скорости передачи больших файлов.

Приложение pipe_example.c:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
    int fd[2];
    // Создаем канал
    if (pipe(fd) == -1) {
        perror("Ошибка создания pipe");
        return 1;
    }
    pid_t pid = fork();
    if (pid == -1) {
        perror("Ошибка создания процесса");
        return 1;
    }

    if (pid == 0) {
        // Первый дочерний процесс пишет в канал
        close(fd[0]);
        char message[] = "Привет через pipe!";
        if (write(fd[1], message, sizeof(message)) == -1) {
            perror("Ошибка записи");
            return 1;
        }
        printf("Отправлено: %s\n", message);
        close(fd[1]);
        return 0;
    }

    pid = fork();
    if (pid == -1) {
        perror("Ошибка создания процесса");
        close(fd[0]);
        close(fd[1]);
        wait(NULL);
        return 1;
    }

    if (pid == 0) {
        // Второй дочерний процесс читает из канала
        close(fd[1]);
        char buffer[100];
        ssize_t n = read(fd[0], buffer, sizeof(buffer) - 1);
        if (n == -1) {
            perror("Ошибка чтения");
            return 1;
        }
        buffer[n] = '\0';
        printf("Получено: %s\n", buffer);
        close(fd[0]);
        return 0;
    }

    // Родитель закрывает канал и ждет оба процесса
    close(fd[0]);
    close(fd[1]);
    wait(NULL);
    wait(NULL);
    return 0;
}
```

Приложение fifo_example.c:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/stat.h>
#include <fcntl.h>

int main() {
    char *fifo = "myfifo";
    // Удаляем старый FIFO, если он остался
    unlink(fifo);
    // Создаем FIFO
    if (mkfifo(fifo, 0666) == -1) {
        perror("Ошибка создания FIFO");
        return 1;
    }
    pid_t pid = fork();

    if (pid == 0) {
        // Первый процесс записывает данные
        int fd = open(fifo, O_WRONLY);
        char message[] = "Привет через FIFO!";
        write(fd, message, sizeof(message));
        printf("Отправлено: %s\n", message);
        close(fd);
        return 0;
    }

    pid = fork();

    if (pid == 0) {
        // Второй процесс читает данные
        int fd = open(fifo, O_RDONLY);
        char buffer[100];
        int n = read(fd, buffer, sizeof(buffer) - 1);
        buffer[n] = '\0';
        printf("Получено: %s\n", buffer);
        close(fd);
        return 0;
    }

    // Родитель ждет завершения процессов
    wait(NULL);
    wait(NULL);
    // Удаляем FIFO
    unlink(fifo);
    return 0;
}
```