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-210БВ-24

Студент: Тодуа Н.Г.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 01.10.25

Москва, 2025

Постановка задачи

Вариант 21.

Родительский процесс создает два дочерних процесса. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с такиv 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.

Общий метод и алгоритм решения

Использованные системные вызовы:

- pid_t fork(void); – создает дочерний процесс.
- int pipe(int *fd); – создаёт канал между двумя процессами. Возвращает массив fd.
  fd[0] — дескриптор для чтения

  fd[1] — дескриптор для записи

- int dup2(int oldfd, int newfd) – Дублирует дескриптор файла, заменяя существующий.
- int execl(const char *path, const char *arg, ...) – Заменяет текущий процесс новым процессом из исполняемого файла.
- pid_t waitpid(pid_t pid, int *status, int options) – Ожидает завершения указанного дочернего процесса.

Межпроцессное взаимодействие реализовано с использованием системных вызовов. Родительский процесс создаёт два дочерних: первый записывает нечётные строки в указанный файл, второй – записывает чётные строки в другой указанный файл. Каждый дочерний процесс инвертирует полученные строки перед записью. Обмен данными между процессами осуществляется через два независимых канала, созданных с помощью системного вызова pipe. Взаимодействие с пользователем происходит исключительно через родительский процесс: он запрашивает имена файлов, принимает входные строки, распределяет их между дочерними процессами и управляет их выполнением.

Код программы

parent.c

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

#define MAX_LINE 4096

int main() {
    char filename1[256], filename2[256];
    printf(format: "Введите название выходного файла для нечетных строк: \n");
    // Ввод имён файлов
    if (!fgets(s: filename1, n: sizeof(filename1), stream: stdin)) {
        fprintf(stream: stderr, format: "Error reading filename1\n");
        exit(status: EXIT_FAILURE);
    }
    filename1[strcspn(s: filename1, reject: "\n")] = 0; // убираем \n

    printf(format: "Введите название выходного файла для нечетных строк: \n");
    if (!fgets(s: filename2, n: sizeof(filename2), stream: stdin)) {
        fprintf(stream: stderr, format: "Error reading filename2\n");
        exit(status: EXIT_FAILURE);
    }
    filename2[strcspn(s: filename2, reject: "\n")] = 0;

    int pipe1[2], pipe2[2];
    if (pipe(pipedes: pipe1) == -1 || pipe(pipedes: pipe2) == -1) {
        perror(s: "pipe");
        exit(status: EXIT_FAILURE);
    }

    // Child 1
    pid_t pid1 = fork();
    if (pid1 == 0) {
        close(fd: pipe1[1]); // закрываем write-конец
        dup2(fd: pipe1[0], fd2: STDIN_FILENO);
        close(fd: pipe1[0]);
        close(fd: pipe2[0]);
        close(fd: pipe2[1]);
        execl(path: "./child", arg: "child", filename1, (char *)NULL);
        perror(s: "execl child1");
        exit(status: EXIT_FAILURE);
    }
```

```c
    pid_t pid2 = fork();
    if (pid2 == 0) {
        close(fd: pipe2[1]);
        dup2(fd: pipe2[0], fd2: STDIN_FILENO);
        close(fd: pipe2[0]);
        close(fd: pipe1[0]);
        close(fd: pipe1[1]);
        execl(path: "./child", arg: "child", filename2, (char *)NULL);
        perror(s: "execl child2");
        exit(status: EXIT_FAILURE);
    }

    // Родитель: закрываем read-концы
    close(fd: pipe1[0]);
    close(fd: pipe2[0]);

    char *line = NULL;
    size_t n = 0;
    ssize_t read;
    int line_number = 0;

    // Читаем строки от пользователя (начиная с 3-й введённой строки!)
    while ((read = getline(lineptr: &line, n: &n, stream: stdin)) != -1) {
        line_number++;

        if (line_number % 2 == 1) {
            // Нечётная → pipe1
            write(fd: pipe1[1], buf: line, n: read);
        } else {
            // Чётная → pipe2
            write(fd: pipe2[1], buf: line, n: read);
        }
    }

    // Закрываем write-концы → дети получат EOF
    close(fd: pipe1[1]);
    close(fd: pipe2[1]);

    // Ждём завершения детей
    waitpid(pid: pid1, stat_loc: NULL, options: 0);
    waitpid(pid: pid2, stat_loc: NULL, options: 0);

    free(ptr: line);
    return 0;
}
```

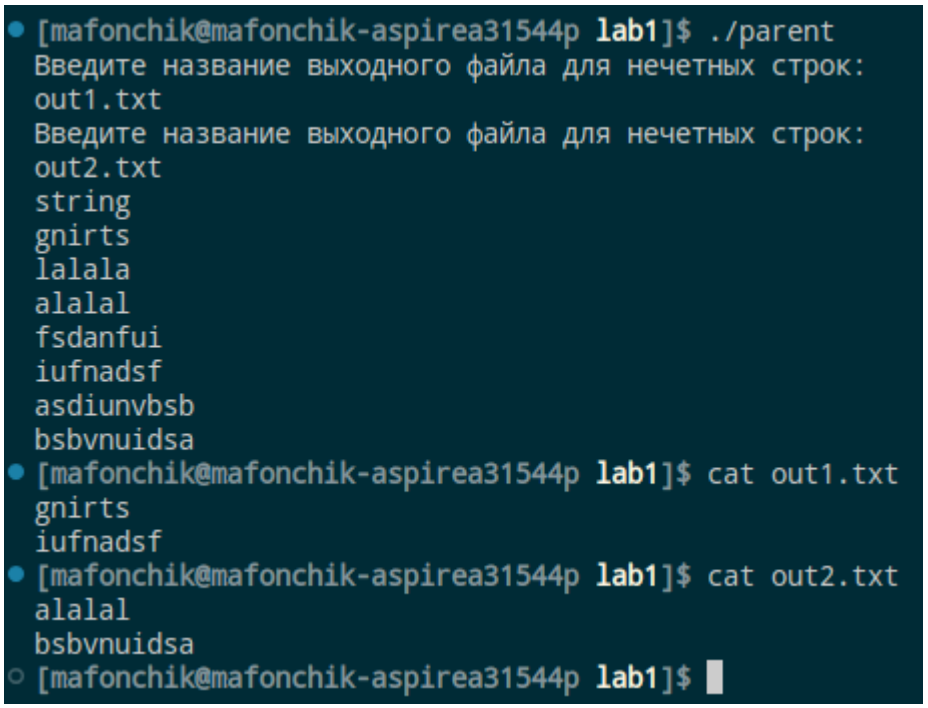
child.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

void reverse_string(char *str) {
    int len = strlen(s: str);
    if (len > 0 && str[len - 1] == '\n') {
        len--; // не включаем \n в реверс
    }
    for (int i = 0; i < len / 2; i++) {
        char tmp = str[i];
        str[i] = str[len - 1 - i];
        str[len - 1 - i] = tmp;
    }
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stream: stderr, format: "Usage: %s <output_file>\n", argv[0]);
        exit(status: EXIT_FAILURE);
    }

    const char *filename = argv[1];
    FILE *fout = fopen(filename, modes: "a"); // append mode
    if (!fout) {
        perror(s: "fopen");
        exit(status: EXIT_FAILURE);
    }

    char *line = NULL;
    size_t n = 0;
    ssize_t read;

    // Читаем из stdin (куда родитель направил pipe)
    while ((read = getline(lineptr: &line, n: &n, stream: stdin)) != -1) {
        // Сохраняем оригинал для вывода в файл и stdout
        char *original = strdup(s: line);
        reverse_string(str: line);

        // Вывод в stdout
        fputs(s: line, stream: stdout);
        fflush(stream: stdout);

        // Запись инвертированной строки в файл
        fputs(s: line, stream: fout);
        fflush(stream: fout);
```

```
        free(ptr: original);
    }

    fclose(stream: fout);
    free(ptr: line);
    return 0;
}
```

Протокол работы программы

Тестирование:



Вывод

В ходе выполнения лабораторной работы было реализовано межпроцессное взаимодействие в Unix-подобной операционной системе с использованием системных вызовов fork, pipe, dup2, exec, waitpid и write. Родительский процесс создаёт два дочерних процесса, каждый из которых получает свой поток данных через отдельный канал, инвертирует полученные строки и записывает результат в соответствующий выходной файл.