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-211БВ-24

Студент: Бурнаев А.А.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 16.10.25

Москва, 2025

Постановка задачи

Вариант 15. Родительский процесс создает дочерний процесс. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child проверяет строки на валидность правилу. Если строка соответствует правилу, то она выводится в стандартный поток вывода дочернего процесса, иначе в pipe2 выводится информация об ошибке. Родительский процесс полученные от child ошибки выводит в стандартный поток вывода

Правило проверки: строка должна начинаться с заглавной буквы

Общий метод и алгоритм решения

Краткое описание программы

Данная лабораторная работа реализует клиент-серверную архитектуру с использованием межпроцессного взаимодействия через pipes. Программа состоит из двух частей: клиентского процесса (первый файл) и серверного процесса (второй файл), которые обмениваются данными через два неименованных канала (pipe).

Использованные системные вызовы

1. pid_t fork(void) – создает дочерний процесс

- Используется для создания отдельного процесса-сервера
- Возвращает:
    - 0 в дочернем процессе
    - PID дочернего процесса в родительском процессе
    - -1 в случае ошибки

2. int pipe(int fd[2]) – создает неименованный канал

- Создает два файловых дескриптора: fd[0] для чтения, fd[1] для записи
- Используется для двусторонней связи между процессами:
    - client_to_server[2] – передача данных от клиента к серверу

  server_to_client[2] – передача ответов от сервера клиенту

3. int dup2(int oldfd, int newfd) – дублирует файловый дескриптор

- В клиенте: dup2(client_to_server[0], STDIN_FILENO) – перенаправляет stdin сервера на чтение из pipe
- В сервере: dup2(file, STDOUT_FILENO) – перенаправляет stdout сервера в файл

4. int execv(const char *path, char *const argv[]) – запускает новую программу

- Заменяет образ текущего процесса образом программы "server"
- Передает аргументы: имя файла и дескриптор для обратной связи

5. ssize_t read(int fd, void *buf, size_t count) – чтение данных

- Чтение имени файла из stdin
- Чтение данных из pipes между процессами
- Чтение пути программы через /proc/self/exe

6. ssize_t write(int fd, const void *buf, size_t count) – запись данных

- Запись данных в pipes
- Вывод сообщений об ошибках
- Запись результатов в файл

7. int readlink(const char *path, char *buf, size_t bufsiz) – чтение символической ссылки

- Используется для определения пути к исполняемому файлу через /proc/self/exe

8. int open(const char *pathname, int flags, mode_t mode) – открытие файла

- Открывает файл для записи с флагами:
    - O_WRONLY – только запись
    - O_CREAT – создать файл если не существует
    - O_TRUNC – очистить файл при открытии
    - O_APPEND – добавлять данные в конец

9. pid_t wait(NULL) – ожидание завершения дочернего процесса

- Обеспечивает корректное завершение работы родительского процесса после сервера

Алгоритм работы программы

Фаза инициализации:

1. Чтение имени файла для вывода из стандартного ввода
2. Определение пути к исполняемому файлу через /proc/self/exe
3. Создание двух pipe для двусторонней связи
4. Создание дочернего процесса через fork()

Работа дочернего процесса (сервер):

1. Перенаправление stdin на чтение из client_to_server
2. Подготовка аргументов для execv

3. Запуск программы "server" с передачей имени файла и дескриптора
4. В серверной программе:
    - Открытие файла для записи
    - Перенаправление stdout в файл
    - Обработка входящих данных

Работа родительского процесса (клиент):

1. Чтение данных из stdin и передача через pipe серверу
2. Получение ответов от сервера через второй pipe
3. Обработка ответов ("0" – ошибка, "1" – успех)
4. Завершение по пустой строке
5. Ожидание завершения дочернего процесса

Логика обработки данных:

- Сервер проверяет, начинается ли строка с заглавной буквы
- Valid строки записываются в файл, клиенту возвращается "1"
- Invalid строки игнорируются, клиенту возвращается "0"

Код программы

client.c

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdint.h>
int main() {
	const char program_name[] = "server";
	char file_name[260];
	ssize_t bytes;
	bytes = read(STDIN_FILENO, file_name, sizeof(file_name));
```

```
if (bytes < 0) {
    const char msg[] = "Failed to read file name\n";
    write(STDOUT_FILENO, msg, sizeof(msg));
    exit(EXIT_FAILURE);
}
file_name[bytes - 1] = '\0';

char progpath[1024];
{
    ssize_t len = readlink("/proc/self/exe", progpath,
        sizeof(progpath) - 1);
        if (len == -1) {
        const char msg[] = "error: failed to read full program path\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }

    while (progpath[len] != '/')
    --len;
    progpath[len] = '\0';
}

int client_to_server[2];
int server_to_client[2];

if (pipe(client_to_server) == -1) {
    const char msg[] = "Failed to create pipe\n";
    write(STDOUT_FILENO, msg, sizeof(msg));
    exit(EXIT_FAILURE);
}
if (pipe(server_to_client) == -1) {
    const char msg[] = "Failed to create pipe\n";
    write(STDOUT_FILENO, msg, sizeof(msg));
```

```
    exit(EXIT_FAILURE);
}

pid_t child = fork();
switch (child) {
    case -1:
    {
        const char msg[] = "error: failed to spawn new process\n";
        write(STDOUT_FILENO, msg, sizeof(msg));
        break;
    }
    case 0:
    {
        close(client_to_server[1]);
        close(server_to_client[0]);

        dup2(client_to_server[0], STDIN_FILENO);
        close(client_to_server[0]);

        char fd[20];
        snprintf(fd, sizeof(fd), "%d", server_to_client[1]);

        char * args[] = {(char *)program_name, file_name, fd, NULL};

        char path[1024];
        snprintf(path, sizeof(path) - 1, "%s/%s", progpath, program_name);

        int32_t status = execv(path, args);
        if (status == -1) {
            const char msg[] = "error: failed to exec into new exectuable image\n";
            write(STDERR_FILENO, msg, sizeof(msg));
            exit(EXIT_FAILURE);
        }
```

```
    break;
}
default:
{
    close(server_to_client[1]);
    close(client_to_server[0]);

    char buffer[4096];
    ssize_t bytes;

    while ((bytes = read(STDIN_FILENO, buffer, sizeof(buffer))) > 0) {
        if (bytes < 0) {
            const char msg[] = "error: failed to read from stdin\n";
            write(STDOUT_FILENO, msg, sizeof(msg));
            exit(EXIT_FAILURE);
        }
        else if (buffer[0] == '\n') {
            break;
        }

        write(client_to_server[1], buffer, bytes);

        bytes = read(server_to_client[0], buffer, sizeof(buffer));
        if (bytes < 0) {
            char msg[] = "Failed to read server\n";
            write(STDERR_FILENO, msg, sizeof(msg));
            break;
        }
        if(buffer[0] == '0') {
            char msg[] = "Wrong string!\n";
            write(STDOUT_FILENO, msg, sizeof(msg));
        }
    }
```

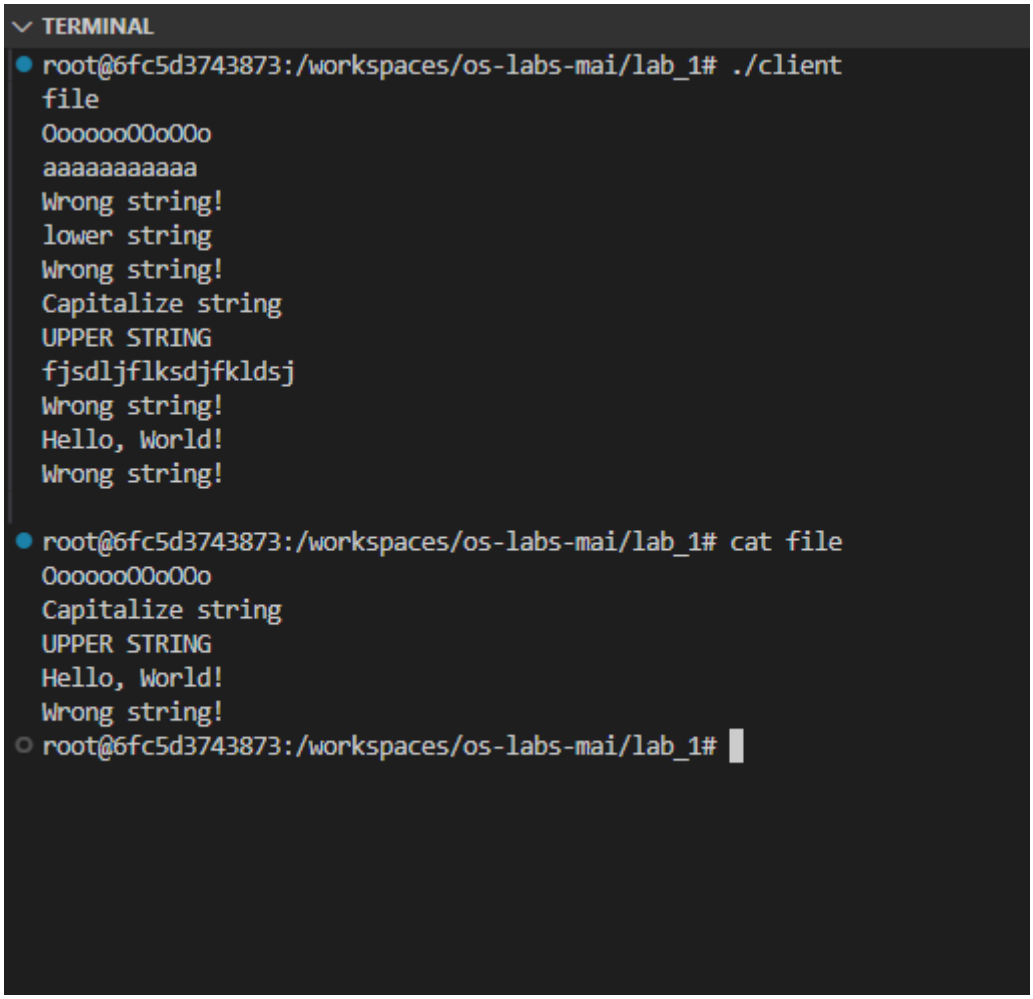
```
            close(client_to_server[1]);
            close(server_to_client[0]);
            wait(NULL);
        }
    }
    exit(EXIT_SUCCESS);
}
```

server.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdint.h>

int main(int argc, const char ** argv) {
    int server_to_client = atoi(argv[2]);
    int32_t file = open(argv[1], O_WRONLY | O_CREAT | O_TRUNC | O_APPEND, 0600);
    if (file == -1) {
        const char msg[] = "error: failed to open requested file\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }

    dup2(file, STDOUT_FILENO);
    close(file);
    int32_t bytes;
    char buffer[1024];
    while ((bytes = read(STDIN_FILENO, buffer, sizeof(buffer))) > 0) {
        if (bytes == -1) {
            const char msg[] = "error: failed to read client\n";
            write(STDERR_FILENO, msg, sizeof(msg));
            exit(EXIT_FAILURE);
```

```
        }


        if (buffer[0] >= 'A' && buffer[0] <= 'Z') {
            write(STDOUT_FILENO, buffer, bytes);
            const char msg[] = "1";
            write(server_to_client, msg, sizeof(msg));
        }
        else {
            const char msg[] = "0";
            write(server_to_client, msg, sizeof(msg));
        }
    }
    close(file);
    return 0;
}
```

Протокол работы программы



Вывод

В ходе выполнения лабораторной работы была успешно реализована клиент-серверная система с использованием межпроцессного взаимодействия через неименованные каналы. Программа продемонстрировала эффективное разделение функциональности между процессами: клиентский процесс отвечает за чтение входных данных и взаимодействие с пользователем, а серверный процесс выполняет валидацию строк и запись результатов в файл. Использование механизма pipes обеспечило надежную двустороннюю связь между процессами.