Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 "Компьютерные науки и прикладная математика"

Кафедра 806 вычислительной математики и программирования

**Лабораторная работа №1 по курсу**

**«Операционные системы»**

**СИСТЕМНЫЕ ВЫЗОВЫ**

Студент: Ильченко Михаил Олегович

Группа: М8О–203БВ–25

Вариант: 22

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_ \_\_\_\_\_

Дата: \_20.10.2025

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва 2025

**Постановка задачи**

## Цель работы

Приобретение практических навыков в:

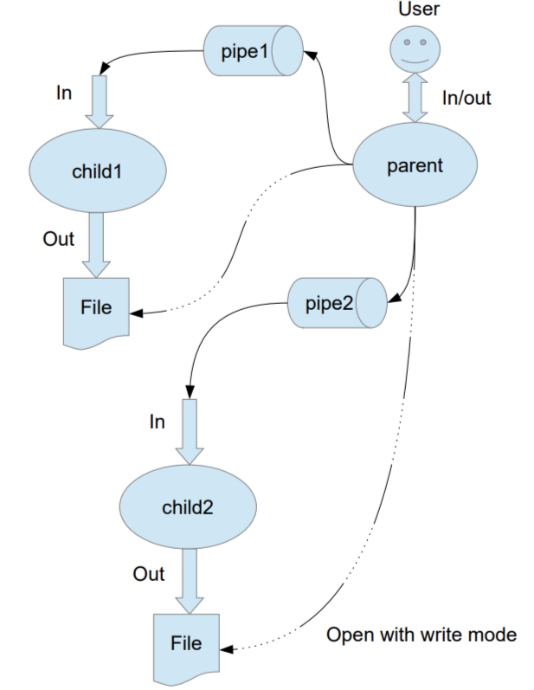
* Управление процессами в ОС
* Обеспечение обмена данных между процессами посредством каналов

## Задание

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.

**Общие сведения о программе**

****

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.

Нужно реализовать правило фильтрации: с вероятностью 80% строки отправляются в pipe1, иначе в pipe2. Дочерние процессы инвертируют строки. Использованы такие системные функции:

• BOOL WINAPI CreateProcess(…) - создание нового процесса

• WaitForSingleObject(…) - ожидание завершения процесса

• CreateFile/SetNamedPipeHandleState - создание именованного канала и установления режима его использования

• OpenFile(…) - открытие нового файла

• CreatePipe(…) - создание безымянного канала

• CreateFile(…) - создание нового файла

• CloseHandle(…) - закрытие объекта ОС по "заголовку". Подходит для закрытия файлов.

**Общий метод и алгоритм решения**.

Для реализации задачи распределения данных между двумя процессами и их параллельной обработки был применён метод **межпроцессного взаимодействия (IPC)** с использованием **безымянных каналов (Pipes)** операционной системы Windows. Весь процесс разделен на два основных компонента: **родительский процесс** (parent.c) и **дочерний процесс** (child.c) .

Родительский процесс выполняет роль **диспетчера и отправителя**. Он с помощью функции CreatePipe создает два независимых канала: $P\_1$ (для дочернего процесса 1) и $P\_2$ (для дочернего процесса 2). Далее, используя структуры STARTUPINFOA, он **перенаправляет стандартный ввод (stdin)** каждого дочернего процесса на **конец для чтения** соответствующего канала. Дочерние процессы, запущенные командой child.exe <filename>, действуют как **получатели и обработчики**. Они ожидают данных на своем стандартном вводе, который фактически является концом канала.

**Алгоритм:**

1. **Инициализация:** Родительский процесс запрашивает имена двух выходных файлов, создает два канала ($P\_1$ и $P\_2$) и запускает два дочерних процесса, передавая им концы для чтения $P\_1$ и $P\_2$ в качестве stdin.
2. **Диспетчеризация:** В главном цикле родительский процесс считывает строки со своего консольного ввода.
3. **Случайное распределение:** Каждая считанная строка с вероятностью 80% записывается в канал $P\_1$ и с вероятностью 20% — в канал $P\_2$, используя функции WriteFile.
4. **Обработка:** Каждый дочерний процесс постоянно читает данные из своего канала, инвертирует каждую полученную строку и записывает результат в свой уникальный выходной файл.
5. **Завершение:** При получении сигнала **EOF** (достижение конца файла) по каналу (что происходит, когда родитель закрывает свой конец для записи, CloseHandle), дочерний процесс завершает работу. Родительский процесс ожидает завершения обоих дочерних процессов с помощью WaitForSingleObject, что гарантирует, что вся обработка завершена до выхода из программы.

**Исходный код можно найти в GitHub:** https://github.com/mikhail0920/OS\_Lab1

**Вывод**

Проведенная лабораторная работа успешно продемонстрировала реализацию базового механизма **межпроцессного взаимодействия (IPC)** в операционной системе Windows с использованием **безымянных каналов (Pipes)** .

Был реализован эффективный клиент-серверный (родитель-дочерний) подход, в котором **родительский процесс** (parent.c) выступал в роли **диспетчера ввода**, а два параллельно запущенных **дочерних процесса** (child.c) — в роли **обработчиков данных**.

Ключевые результаты и освоенные концепции:

1. **Создание и перенаправление каналов (Pipes):** С помощью функций CreatePipe и структур SECURITY\_ATTRIBUTES и STARTUPINFOA удалось создать два независимых однонаправленных канала и успешно перенаправить стандартный ввод (stdin) каждого дочернего процесса на конец чтения соответствующего канала.
2. **Параллельная обработка:** Использование функции CreateProcessA позволило запустить два дочерних процесса одновременно, обеспечив **параллельную обработку** поступающих данных (инвертирование строк), что является основой многозадачности.
3. **Управление ресурсами и EOF:** Была отработана критически важная процедура управления дескрипторами: родительский процесс закрывал свои концы чтения канала сразу после создания дочерних процессов (CloseHandle(p\*\_read)), а концы записи закрывались в конце работы (CloseHandle(p\*\_write)) для корректной передачи **сигнала EOF** дочерним процессам. Это гарантировало, что дочерние процессы не будут бесконечно ожидать ввода и корректно завершат свою работу.
4. **Синхронизация:** Применение функции WaitForSingleObject обеспечило **синхронизацию** родительского процесса с дочерними, гарантируя, что родитель завершится только после того, как оба дочерних процесса закончат свою работу и запишут все результаты в файлы.

Таким образом, лабораторная работа подтвердила глубокое понимание принципов работы Windows API для создания и управления процессами, организации однонаправленного потока данных через каналы, а также методов синхронизации и завершения процессов.