**Московский авиационный институт (национальный исследовательский университет)**

Институт информационных технологий и прикладной математики

«Кафедра вычислительной математики и программирования»

**Лабораторная работа по предмету "Операционные системы" №1**

Студент: Жидких Д.С.

Преподаватель: Миронов Е.С.

Группа: М8О-201БВ-24

Оценка:\_\_\_\_\_\_\_\_\_

Дата:\_\_\_\_\_\_\_\_\_

Подпись:\_\_\_\_\_\_\_\_\_

**Оглавление**

1. Цель работы
2. Постановка задачи
3. Общие сведения о программе
4. Общий алгоритм решения
5. Реализация
6. Пример работы
7. Вывод

# 

# **Цель работы**

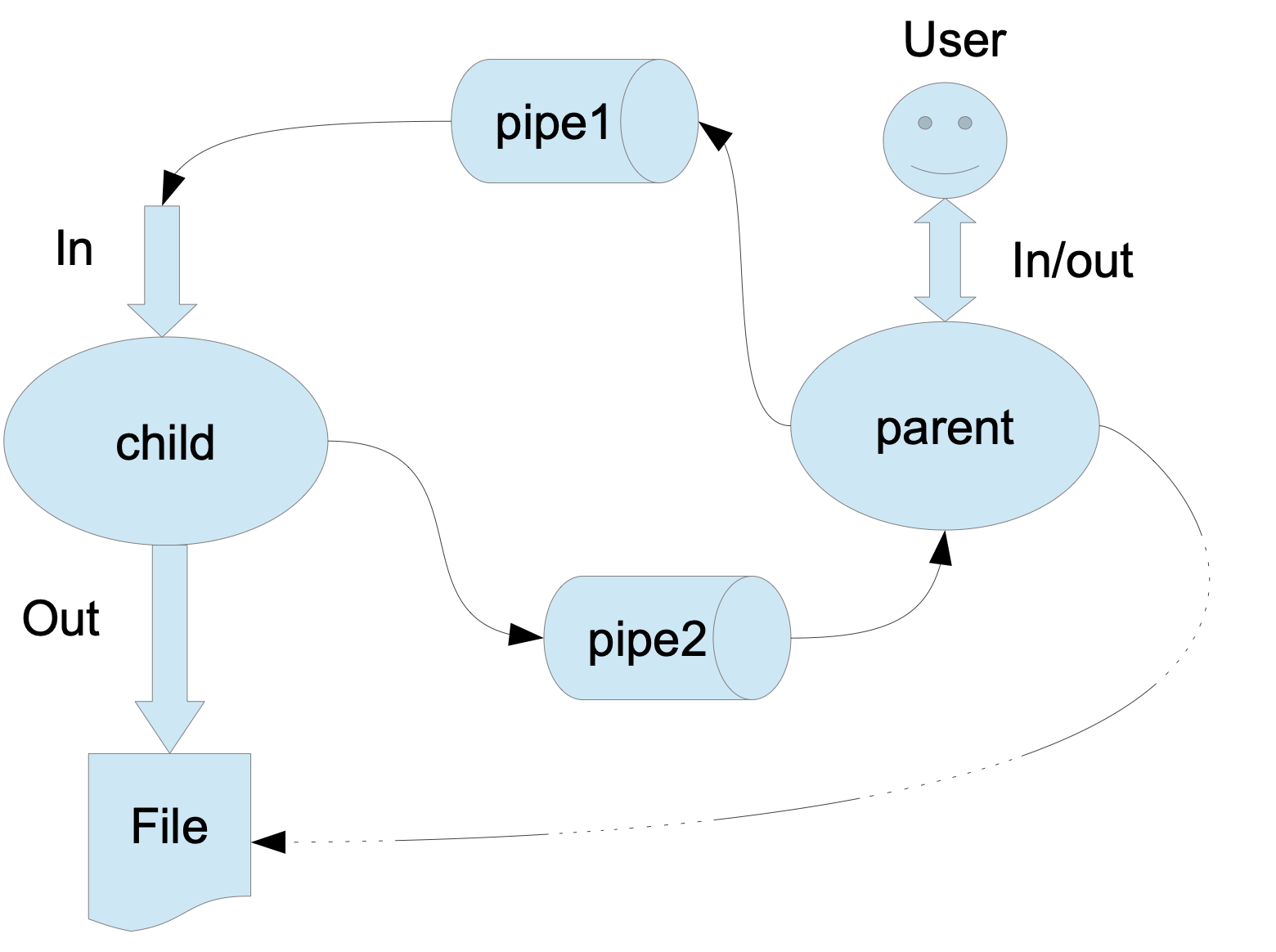
Приобретение практических навыков в:

* Управлении процессами в ОС
* Обеспечении обмена данных между процессами посредством каналов

# **Постановка задачи**

Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате ё 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 4**

**

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.

# **Общие сведения о программе**

Программа представлена двумя файлами:

* parent.c — программа-родитель;
* child.c — программа-ребёнок.

Используемые системные вызовы:

* pid\_t fork() — создание дочернего процесса;
* int execl(const char \*path, const char \*arg0, ..., NULL) — замена текущего образа процесса другой программой;
* pid\_t wait(int \*stat\_loc) — ожидание завершения дочернего процесса;
* int pipe(int pipefd[2]) — создание неименованного канала;
* int dup2(int oldfd, int newfd) — переназначение файловых дескрипторов;
* int open(const char \*pathname, int flags, mode\_t mode) — открытие/создание файла;
* int close(int fd) — закрытие файлового дескриптора;

# 

# **Общий алгоритм решения**

1. Родительский процесс создаёт два канала: pipe1 (родитель → ребёнок) и pipe2 (ребёнок → родитель).
2. Родительский процесс выполняет fork().
3. В дочернем процессе:
   * переназначается stdin на pipe1[0],
   * переназначается stdout на pipe2[1],
   * запускается программа child через execl.
4. В родительском процессе:
   * пользователь вводит строки,
   * родитель пересылает их в pipe1[1],
   * родитель параллельно получает из pipe2[0] сообщения об ошибках и печатает их.
5. В дочернем процессе (child.c):
   * первая строка — имя файла,
   * последующие строки проверяются на окончание символами . или ;,
   * корректные строки записываются в файл,
   * ошибки передаются через stdout (который связан с pipe2).
6. Родитель ждёт завершения ребёнка (wait).

# **Пример работы**

**Ввод пользователя (через родителя):**

out.txt

Hello world.

Строка без конца

Another one;

Done.

**Содержимое файла out.txt:**

Hello world.

Another one;

Done.

**Вывод на экран родителя:**

Error: строка должна заканчиваться на '.' или ';' - 'Строка без конца'

# **Вывод**

В ходе выполнения лабораторной работы были получены практические навыки:

* создания и использования неименованных каналов для обмена данными между процессами;
* перенаправления стандартных потоков ввода-вывода с помощью системного вызова dup2;
* управления процессами с использованием fork, execl и wait;
* организации взаимодействия между двумя независимыми программами.

Реализованная программа демонстрирует базовые приёмы межпроцессного взаимодействия и перенаправления потоков, которые применяются в операционных системах UNIX-подобного типа.