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»

Выполнила: Власова Е.Р.
Группа: М8О-208БВ-24
Преподаватель: А. Ядров

Москва, 2025

Условие:

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.

Цель работы:

Приобретение практических навыков в управлении процессов в ОС, Обеспечение обмена данных между процессами посредством каналов.

Задание:

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.

Вариант:

15

Метод решения:

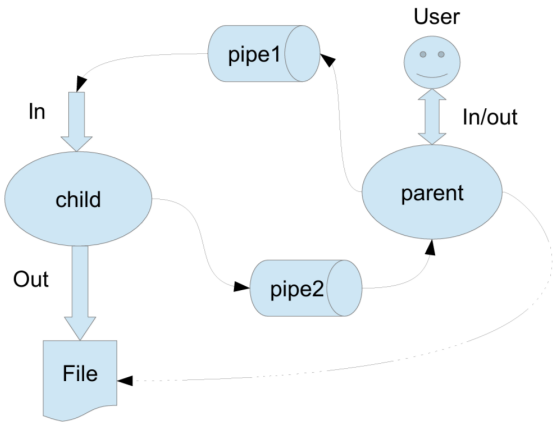
Программа представляет собой систему межпроцессного взаимодействия для фильтрации текстовых строк. Родительский процесс принимает строки от пользователя, передает их дочернему процессу для проверки, и сохраняет результаты в файл.

Описание программы:

Программа состоит из следующих файлов: common.h, common.cpp, child.cpp, parent.cpp.

Я использую следующие системные вызовы(для Linux, аналогичные для Windows): read() write() open() close() pipe()

Результаты:



Выводы:

Приобрела практические навыкы в управлении процессов в ОС и узнала про обеспечение обмена данных между процессами посредством каналов. Осознала суть понятий канала и процесса, поняла границы их применения в рамках ОС. Выяснила как можно создавать pipe, перенаправлять с их помощью выводы для процессов, как осуществлять создание дочерних процессов и какая между ними разница.

Исходная программа:

Файл common.h

```
#ifndef COMMON_H
#define COMMON_H
#include <string>

const std::string INVALID_STRING = "Error: string does not
start with uppercase letter\n";
const std::string FILE_INPUT = "Enter filename: ";
const std::string STRING_INPUT = "Enter a string: ";
const std::string FILE_ERROR = "Error opening file\n";

const int RD = 0;
const int WR = 1;

int readString(int fd, std::string &line);
void writeString(int fd, const std::string &line);
bool startsWithUppercase(const std::string &str);

#endif
```

Файл common.cpp

```
#include "common.h"
#include <cctype>
#ifdef _WIN32
#include <io.h>
#include <windows.h>
#else
#include <unistd.h>
#endif

int readString(int fd, std::string &line) {
    char symbol;
    line.clear();
    while (true) {
        int res;
#ifdef _WIN32
        res = _read(fd, &symbol, sizeof(char));
#else
```

```
        res = read(fd, &symbol, sizeof(char));
#endif
        if (res <= 0) {
            return EOF;
        }

        if (symbol == '\n') {
            break;
        }

        line.push_back(symbol);
    }

    return line.length();
}

void writeString(int fd, const std::string &line) {
#ifdef _WIN32
    _write(fd, line.c_str(), line.length());
#else
    write(fd, line.c_str(), line.length());
#endif
}

bool startsWithUppercase(const std::string &str) {
    if (str.empty()) {
        return false;
    }
    return std::isupper(static_cast<unsigned char>(str[0]));
}
```

Файл child.cpp

```
#include <string>
#ifdef _WIN32
#include <io.h>
#include <windows.h>
#define STDIN_FILENO 0
#define STDOUT_FILENO 1
#define STDERR_FILENO 2
#else
```

```
#include <unistd.h>
#endif

#include "common.h"

int main() {
    std::string stringLine;

    while (readString(STDIN_FILENO, stringLine) != EOF) {
        if (startsWithUppercase(stringLine)) {
            writeString(STDOUT_FILENO, stringLine + "\n");
        } else {
            writeString(STDERR_FILENO, INVALID_STRING);
        }
        stringLine.clear();
    }
    return 0;
}
```

Файл parent.cpp

```
#include <iostream>
#include <string>
#include <cstdlib>

#ifdef _WIN32
#include <io.h>
#include <windows.h>
#include <fcntl.h>
#define STDIN_FILENO 0
#define STDOUT_FILENO 1
#define STDERR_FILENO 2
#define pipe(fds) _pipe(fds, 4096, _O_BINARY)
#define close _close
#else
#include <unistd.h>
#include <fcntl.h>
#include <sys/wait.h>
#endif

#include "common.h"
```

```
int main() {
    int pipe1[2];
    int pipe2[2];

    if (pipe(pipe1) == -1) {
        perror("pipe");
        exit(EXIT_FAILURE);
    }

    if (pipe(pipe2) == -1) {
        perror("pipe");
        exit(EXIT_FAILURE);
    }

    writeString(STDOUT_FILENO, FILE_INPUT);

    std::string filename;
    if (readString(STDIN_FILENO, filename) == EOF) {
        writeString(STDOUT_FILENO, FILE_ERROR);
        exit(EXIT_FAILURE);
    }

    #ifdef _WIN32
        int file = _open(filename.c_str(), _O_CREAT |
        _O_WRONLY | _O_APPEND, 0644);
    #else
        int file = open(filename.c_str(),
        O_CREAT | O_WRONLY | O_APPEND, S_IRUSR | S_IWUSR);
    #endif

        if (file < 0) {
            writeString(STDOUT_FILENO, FILE_ERROR);
            perror("file");
            exit(EXIT_FAILURE);
        }

    #ifdef _WIN32
    STARTUPINFO si;
    PROCESS_INFORMATION pi;
    ZeroMemory(&si, sizeof(si));
```

```
si.cb = sizeof(si);
ZeroMemory(&pi, sizeof(pi));

si.dwFlags |= STARTF_USESTDHANDLES;
si.hStdInput = (HANDLE)_get_osfhandle(pipe1[RD]);
si.hStdOutput = (HANDLE)_get_osfhandle(pipe2[WR]);
si.hStdError = (HANDLE)_get_osfhandle(pipe2[WR]);

std::string command = "child.exe";
if (!CreateProcess(NULL, (LPSTR)command.c_str(),
NULL, NULL, TRUE, 0, NULL, NULL, &si, &pi)) {
    std::cerr << "CreateProcess failed: " <<
    GetLastError() << std::endl;
    exit(EXIT_FAILURE);
}

close(pipe1[RD]);
close(pipe2[WR]);

while (true) {
    writeString(STDOUT_FILENO, STRING_INPUT);

    std::string str;
    if (readString(STDIN_FILENO, str) == EOF) {
        close(pipe1[WR]);
        break;
    }
    writeString(pipe1[WR], str + "\n");

    std::string response;
    readString(pipe2[RD], response);
    writeString(file, response);
}
    WaitForSingleObject(pi.hProcess, INFINITE);
    CloseHandle(pi.hProcess);
    CloseHandle(pi.hThread);

#else
    pid_t pid = fork();

    if (pid < 0) {
```

```
            perror("fork");
            exit(EXIT_FAILURE);
        }

        if (pid == 0) {
            close(pipe2[RD]);
            close(pipe1[WR]);

            dup2(pipe1[RD], STDIN_FILENO);
            dup2(pipe2[WR], STDOUT_FILENO);
            dup2(pipe2[WR], STDERR_FILENO);

            execl("./child", "./child", NULL);
            perror("execl");
            exit(EXIT_FAILURE);
        } else {
            close(pipe1[RD]);
            close(pipe2[WR]);

            while (true) {
                writeString(STDOUT_FILENO, STRING_INPUT);
                std::string str;
                if (readString(STDIN_FILENO, str) == EOF) {
                    close(pipe1[WR]);
                    break;
                }
                writeString(pipe1[WR], str + "\n");

                std::string response;
                readString(pipe2[RD], response);
                writeString(file, response);
            }

            wait(NULL);
            close(pipe2[RD]);
        }
    #endif

    close(file);
    return 0;
}
```