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-203Б-23

Студент: Никитцев А.В.

Преподаватель: Миронов Е.С.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: 25.12.24

Москва, 2024

**Цель работы**

Приобретение практических навыков в:

• Управление процессами в ОС

• Обеспечение обмена данных между процессами посредством каналов

**Постановка задачи (Вариант 17)**

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.

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.

Правило фильтрации: строки длины больше 10 символов отправляются в pipe2, иначе в pipe1. Дочерние процессы удаляют все гласные из строк.

**Код**

**Child1.cpp**

#include <string>  
#include <csignal>  
#include <iostream>  
#include "utils.h"  
  
int main(int argv, char \*\* argc) {  
 FILE \* file = fopen(argc[0], "w");  
 dup2(fileno(file), 1);  
  
 ReadData([](const std::string& str){  
 std::string res = Modify(str);  
 write(1, res.c\_str(), res.size());  
 }, std::cin);  
  
 fclose(file);  
 close(1);  
}

**Child2.cpp**

#include <string>  
#include <csignal>  
#include <iostream>  
#include "utils.h"  
  
int main(int argv, char \*\* argc){  
 FILE \* file = fopen(argc[0], "w");  
 dup2(fileno(file), 1);  
  
 ReadData([](const std::string& str){  
 std::string res = Modify(str);  
 write(1, res.c\_str(), res.size());  
 }, std::cin);  
  
 fclose(file);  
 close(1);  
}

**Constants.h**

#pragma once  
  
#include <string>  
  
const std::string CHILD\_1\_PATH {"./child1"};  
const std::string CHILD\_2\_PATH {"./child2"};  
const int READ\_END = 0;  
const int WRITE\_END = 1;  
const int ERROR = -1;  
const int MAX\_STRING\_LENGTH = 10;  
const int CHILD\_PROCESS = 0;

**Lib.cpp**

#include <iostream>  
#include "lib.h"  
#include <cstdio>  
#include <cstdlib>  
#include "unistd.h"  
#include "utils.h"  
#include "constants.h"  
#include <sys/wait.h>  
#include <fstream>  
  
std::string GetFileName(std::istream& stream){  
 std::string buff;  
 std::getline(stream, buff);  
 return buff;  
}  
  
bool StartProcess(int \* pipe, const std::string& childPath, std::string& filePath) {  
 pid\_t pid = fork();  
  
 if (pid == ERROR){  
 perror("Can't fork process");  
 exit(ERROR);  
 }  
  
 if (pid == CHILD\_PROCESS){  
 close(pipe[WRITE\_END]);  
 dup2(pipe[READ\_END], READ\_END);  
  
 if (execl(childPath.c\_str(), const\_cast<char \*>(filePath.c\_str()), NULL) == ERROR){  
 std::cout << "Something went wrong when creating process " << childPath << std::endl;  
 }  
 }  
  
 return pid == CHILD\_PROCESS;  
}  
  
void ParentMain(std::istream& input){  
 int pipe\_to\_child\_1[2];  
 OpenPipe(pipe\_to\_child\_1);  
  
 int pipe\_to\_child\_2[2];  
 OpenPipe(pipe\_to\_child\_2);  
  
 std::string fName1, fName2;  
  
 std::cout << "Enter filename for 1 process: " << std::endl;  
 fName1 = GetFileName(input);  
  
 std::cout << "Enter filename for 2 process: " << std::endl;  
 fName2 = GetFileName(input);  
  
 std::cout << "FILE NAMES: "<< fName1 << " " << fName2 << std::endl;  
  
 if (StartProcess(pipe\_to\_child\_2, CHILD\_2\_PATH, fName2)) return;  
 if (StartProcess(pipe\_to\_child\_1, CHILD\_1\_PATH, fName1)) return;  
  
 std::cout << "Enter strings to process: " << std::endl;  
  
 ReadData([pipe\_to\_child\_1, pipe\_to\_child\_2](const std::string& str) {  
 if (str.length() > MAX\_STRING\_LENGTH) write(pipe\_to\_child\_2[WRITE\_END], str.c\_str(), str.size());  
 else write(pipe\_to\_child\_1[WRITE\_END], str.c\_str(), str.size());  
 }, input);  
  
 write(pipe\_to\_child\_1[WRITE\_END], "\n", 1);  
 write(pipe\_to\_child\_2[WRITE\_END], "\n", 1);  
  
 close(pipe\_to\_child\_2[READ\_END]);  
 close(pipe\_to\_child\_1[READ\_END]);  
  
 close(pipe\_to\_child\_2[WRITE\_END]);  
 close(pipe\_to\_child\_1[WRITE\_END]);  
  
 wait(nullptr);  
 wait(nullptr);  
}

**Lib.h**

#pragma once  
  
#include <fstream>  
  
void ParentMain(std::basic\_istream<char>& input);

**Utils.cpp**

#include <string>  
#include <vector>  
#include <csignal>  
#include <fstream>  
#include <iostream>  
  
#include "utils.h"  
#include "constants.h"  
  
const std::string vowels = "aeiouy";  
  
void ReadData(const std::function<void(const std::string&)>& handler, std::basic\_istream<char>& stream){  
 std::string buff;  
  
 while (std::getline(stream, buff)){  
 if (buff.empty()){  
 return;  
 }  
 handler(buff + '\n');  
 }  
}  
  
std::string Modify(const std::string& str){  
 std::string res;  
  
 for (auto& c : str){  
 if (vowels.find(c) == ERROR){  
 res += c;  
 }  
 }  
  
 return res;  
}  
  
void OpenPipe(int \*p) {  
 if (pipe(p) == ERROR){  
 perror("Can't open pipe");  
 exit(ERROR);  
 }  
}

**Utils.h**

#pragma once  
  
#include <string>  
#include <vector>  
#include <functional>  
  
void ReadData(const std::function<void(const std::string&)>& handler, std::basic\_istream<char>& stream);  
std::string Modify(const std::string& str);  
void OpenPipe(int \*p);

**Test.cpp**

#include <gtest/gtest.h>  
#include <fstream>  
#include <filesystem>  
#include "../src/utils.h"  
#include "../src/lib.h"  
  
const char \*const INPUT\_FILE\_NAME = "./input.txt";  
  
std::string Read(std::ifstream& file){  
 std::string buff;  
 std::string result;  
  
 while (std::getline(file, buff)){  
 result += buff + '\n';  
 }  
  
 return result;  
}  
  
void RemoveIfExists(const std::string& filename){  
 if (std::filesystem::exists(filename)) std::filesystem::remove(filename);  
}  
  
TEST(Should\_RemoveAllVowels\_When\_StringContainsVowels, utils\_test){  
 std::string str = "qweyiaeiouywwwwwtplk";  
 ASSERT\_EQ("qwwwwwwtplk", Modify(str));  
}  
  
TEST(Should\_ReturnTheSameString\_When\_StringDoesNotContainVowels, utils\_test){  
 std::string str = "qwwwwwwtplk";  
 ASSERT\_EQ("qwwwwwwtplk", Modify(str));  
}  
  
TEST(Should\_WriteToRightFile, main\_test){  
 std::array<std::string, 8> data = {  
 "file1.txt",  
 "file2.txt",  
 "qwerty",  
 "qqqqqqqqqqwerty",  
 "pipe",  
 "very long woooord",  
 "many long words here",  
 "\n"  
 };  
  
 std::string expectedResult2 = "qqqqqqqqqqwrt\nvr lng wrd\nmn lng wrds hr\n";  
 std::string expectedResult1 = "qwrt\npp\n";  
  
 FILE\* iFile = fopen(INPUT\_FILE\_NAME, "w");  
  
 for (auto &item : data) {  
 write(fileno(iFile), (item + '\n').c\_str(), item.size() + 1);  
 }  
  
 fclose(iFile);  
  
 std::ifstream inputFile(INPUT\_FILE\_NAME);  
  
 ParentMain(inputFile);  
  
 std::ifstream file1(data[0]);  
 std::ifstream file2(data[1]);  
  
 ASSERT\_TRUE(file1.good());  
 ASSERT\_TRUE(file2.good());  
  
 std::string result1 = Read(file1);  
 std::string result2 = Read(file2);  
  
 ASSERT\_EQ(result1, expectedResult1);  
 ASSERT\_EQ(result2, expectedResult2);  
  
 RemoveIfExists(INPUT\_FILE\_NAME);  
 RemoveIfExists(data[0]);  
 RemoveIfExists(data[1]);  
}  
  
int main(int argc, char \*\*argv) {  
 testing::InitGoogleTest(&argc, argv);  
 return RUN\_ALL\_TESTS();  
}

**Main.h**

#include <iostream>  
#include "src/lib.h"  
  
int main(int argv, char\*\* argc){  
 ParentMain(std::cin);  
}

**Заключение**

Выполнив лабораторную работу, я приобрел навыки управления процессами и отправления данных между ними.