Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

**Лабораторная работа №3 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:  
• Освоение принципов работы с файловыми системами  
• Обеспечение обмена данных между процессами посредством технологии «File mapping»

**Постановка задачи (Вариант 30)**

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 через отображаемые файлы (memory-mapped files).  
Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

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.

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 "utils.h"  
#include "constants.h"  
#include "SharedObjectHandler.h"  
  
int main(int argv, char \*\* argc) {  
 auto handlerChild = SharedObjectHandler(CHILD\_1\_OBJECT\_NAME, CHILD\_1\_OBJECT\_NAME);  
  
 FILE \* file = fopen(argc[0], "w");  
 if (!file){  
 perror("Can't open file");  
 return -1;  
 }  
  
 if (dup2(fileno(file), 1) == -1){  
 perror("Can't redirect output");  
 return -1;  
 }  
  
 std::string data;  
  
 while (true){  
 handlerChild.LockChild();  
  
 data = handlerChild.Read();  
 if (data == "\n") {  
 handlerChild.UnlockChild();  
 handlerChild.UnlockParent();  
 break;  
 }  
  
 std::string res = Modify(data);  
 write(1, res.c\_str(), res.size());  
  
 handlerChild.UnlockParent();  
 }  
  
 fclose(file);  
}

**Child2.cpp**

#include <string>  
#include <csignal>  
#include "utils.h"  
#include "constants.h"  
#include "SharedObjectHandler.h"  
  
int main(int argv, char \*\* argc){  
 auto handlerChild = SharedObjectHandler(CHILD\_2\_OBJECT\_NAME, CHILD\_2\_OBJECT\_NAME);  
  
 FILE \* file = fopen(argc[0], "w");  
 if (!file){  
 perror("Can't open file");  
 return -1;  
 }  
  
 if (dup2(fileno(file), 1) == -1){  
 perror("Can't redirect output");  
 return -1;  
 }  
  
 std::string data;  
  
 while (true){  
 handlerChild.LockChild();  
  
 data = handlerChild.Read();  
 if (data == "\n") {  
 handlerChild.UnlockChild();  
 handlerChild.UnlockParent();  
 break;  
 }  
  
 std::string res = Modify(data);  
 write(1, res.c\_str(), res.size());  
  
 handlerChild.UnlockParent();  
 }  
  
 fclose(file);  
}

**Constants.h**

#pragma once  
  
#include <string>  
  
const inline std::string CHILD\_1\_PATH {"./child1"};  
const inline std::string CHILD\_2\_PATH {"./child2"};  
const inline std::string PARENT\_SEM\_NAME {"parent"};  
const inline std::string CHILD\_1\_OBJECT\_NAME {"/child1"};  
const inline std::string CHILD\_2\_OBJECT\_NAME {"/child2"};  
const constexpr int READ\_END = 0;  
const constexpr int WRITE\_END = 1;  
const constexpr int ERROR = -1;  
const constexpr int MAX\_STRING\_LENGTH = 10;  
const constexpr int CHILD\_PROCESS = 0;  
const constexpr size\_t BUFFER\_SIZE = 4096;

**Lib.cpp**

#include <iostream>  
#include "lib.h"  
#include <cstdio>  
#include <cstdlib>  
#include "unistd.h"  
#include "utils.h"  
#include "constants.h"  
#include "SharedObjectHandler.h"  
#include <sys/wait.h>  
#include <sys/mman.h>  
  
std::string GetFileName(std::istream& stream){  
 std::string buff;  
 std::getline(stream, buff);  
 return buff;  
}  
  
bool StartProcess(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){  
 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){  
 auto handlerChild1 = SharedObjectHandler(CHILD\_1\_OBJECT\_NAME, CHILD\_1\_OBJECT\_NAME, O\_RDWR, PROT\_WRITE, false);  
 auto handlerChild2 = SharedObjectHandler(CHILD\_2\_OBJECT\_NAME, CHILD\_2\_OBJECT\_NAME, O\_RDWR, PROT\_WRITE, false);  
  
  
 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);  
  
 if (StartProcess(CHILD\_2\_PATH, fName2)) return;  
 if (StartProcess(CHILD\_1\_PATH, fName1)) return;  
  
 std::cout << "Enter strings to process: " << std::endl;  
  
 std::string buff;  
  
 while (std::getline(input, buff)){  
 if (buff.empty()) {  
 handlerChild2.Write("\n");  
 handlerChild2.UnlockChild();  
 handlerChild2.LockParent();  
 handlerChild1.Write("\n");  
 handlerChild1.UnlockChild();  
 handlerChild1.LockParent();  
 break;  
 }  
  
 if (buff.length() > MAX\_STRING\_LENGTH) {  
 handlerChild2.Write(buff + '\n');  
 handlerChild2.UnlockChild();  
 handlerChild2.LockParent();  
 }  
 else {  
 handlerChild1.Write(buff + '\n');  
 handlerChild1.UnlockChild();  
 handlerChild1.LockParent();  
 }  
 }  
  
 wait(nullptr);  
 wait(nullptr);  
}

**Lib.h**

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

**SharedObjectHandler.cpp**

#include "SharedObjectHandler.h"  
  
int OpenSharedMemory(const std::string& name, bool exists, mode\_t mode, size\_t size) {  
 auto result = shm\_open(name.c\_str(), mode | O\_CREAT, 0777);  
 if (result == -1) perror("Can't create/read shared memory object");  
  
 if (exists) return result;  
  
 if (ftruncate(result, size) == -1) perror("Can't resize shared memory object");  
 return result;  
}  
  
SharedObjectHandler::SharedObjectHandler(  
 const std::string &sNameChild,  
 const std::string &objName,  
 const mode\_t fdMode,  
 const int prot,  
 const bool exists,  
 const size\_t sz  
): bufferSize(sz){  
 auto fd = OpenSharedMemory(objName, exists, fdMode, sz);  
 buffer = static\_cast<char\*>(mmap(nullptr, sz, prot, MAP\_SHARED, fd, 0));  
 if (buffer == (char\*)(-1)) perror("Error while creating buffer.");  
 semNameChild = sNameChild;  
 semNameParent = sNameChild + "\_parent";  
 objectName = objName;  
 semChild = sem\_open(semNameChild.c\_str(), O\_CREAT, 0666 , 0);  
 semParent = sem\_open(semNameParent.c\_str(), O\_CREAT, 0666 , 0);  
 if (semChild == SEM\_FAILED || semParent == SEM\_FAILED) perror("Some semaphores can't be constructed");  
}  
  
std::string SharedObjectHandler::Read() {  
 std::string buff {buffer};  
 return buff;  
}  
  
void SharedObjectHandler::Write(const std::string &data) {  
 if (data.size() >= bufferSize) perror("Buffer overflow");  
 strcpy(buffer, data.c\_str());  
}  
  
void SharedObjectHandler::LockParent() { sem\_wait(semParent); }  
void SharedObjectHandler::LockChild() { sem\_wait(semChild); }  
void SharedObjectHandler::UnlockParent() { sem\_post(semParent); }  
void SharedObjectHandler::UnlockChild() { sem\_post(semChild); }  
  
SharedObjectHandler::~SharedObjectHandler() {  
 munmap(buffer, bufferSize);  
 shm\_unlink(objectName.c\_str());  
 sem\_unlink(semNameParent.c\_str());  
 sem\_unlink(semNameChild.c\_str());  
}

**SharedObjectHandler.h**

#pragma once  
  
#include <iostream>  
#include "unistd.h"  
#include "constants.h"  
#include <sys/mman.h>  
#include <fcntl.h>  
#include <semaphore>  
#include <functional>  
#include <cstring>  
  
class SharedObjectHandler {  
 sem\_t\* semChild;  
 sem\_t\* semParent;  
 char\* buffer;  
 const size\_t bufferSize;  
 std::string semNameChild, semNameParent, objectName;  
  
 public:  
 SharedObjectHandler(  
 const std::string& sNameChild,  
 const std::string& objName,  
 const mode\_t fdMode = O\_RDONLY,  
 const int prot = PROT\_READ,  
 bool exits = true,  
 size\_t sz = BUFFER\_SIZE  
 );  
  
 std::string Read();  
  
 void Write(const std::string& data);  
  
 void LockParent();  
 void LockChild();  
  
 void UnlockParent();  
 void UnlockChild();  
  
 ~SharedObjectHandler();  
};

**Utils.cpp**

#include <string>  
#include <vector>  
#include <csignal>  
#include <fstream>  
#include <iostream>  
  
#include "utils.h"  
#include "constants.h"  
  
const std::string vowels = "aeiouy";  
  
std::string Modify(const std::string& str){  
 std::string res;  
  
 for (auto& c : str){  
 if (vowels.find(c) == std::string::npos){  
 res += c;  
 }  
 }  
  
 return res;  
}

**Utils.h**

#pragma once  
  
#include <string>  
#include <vector>  
#include <functional>  
  
std::string Modify(const std::string& str);

**Test.cpp**

#include <gtest/gtest.h>  
#include <fstream>  
#include <filesystem>  
#include "../src/utils.h"  
#include "../src/lib.h"  
  
const char \*const INPUT\_FILE\_NAME = "./input1.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(UtilsTest, ShouldRemoveAllVowelsWhenStringContainsVowels){  
 std::string str = "qweyiaeiouywwwwwtplk";  
 EXPECT\_EQ("qwwwwwwtplk", Modify(str));  
}  
  
TEST(UtilsTest, ShouldReturnTheSameStringWhenStringDoesNotContainVowels){  
 const std::string str = "qwwwwwwtplk";  
 EXPECT\_EQ("qwwwwwwtplk", Modify(str));  
}  
  
TEST(Should\_WriteToRightFile, main\_test){  
 const std::array<std::string, 8> data = {  
 "file1.txt\n",  
 "file2.txt\n",  
 "qwerty\n",  
 "qqqqqqqqqqwerty\n",  
 "pipe\n",  
 "very long woooord\n",  
 "many long words here\n",  
 "\n"  
 };  
  
 const std::string expectedResult2 = "qqqqqqqqqqwrt\nvr lng wrd\nmn lng wrds hr\n";  
 const std::string expectedResult1 = "qwrt\npp\n";  
  
 FILE\* iFile = fopen(INPUT\_FILE\_NAME, "w");  
  
 for (const auto &item : data) {  
 write(fileno(iFile), (item).c\_str(), item.size());  
 }  
  
 fclose(iFile);  
  
 std::ifstream inputFile(INPUT\_FILE\_NAME);  
  
 ParentMain(inputFile);  
  
 const std::string fileName1 = "file1.txt";  
 const std::string fileName2 = "file2.txt";  
  
 std::ifstream file1(fileName1);  
 std::ifstream file2(fileName2);  
  
 EXPECT\_TRUE(file1.good());  
 EXPECT\_TRUE(file2.good());  
  
 const std::string result1 = Read(file1);  
 const std::string result2 = Read(file2);  
  
 EXPECT\_EQ(result1, expectedResult1);  
 EXPECT\_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.cpp**

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

**Заключение**

В ходе выполнения работы были успешно приобретены практические навыки работы с файловыми системами и межпроцессным взаимодействием посредством технологии «File mapping». Разработанная программа на языке Си эффективно создает родительский процесс и два дочерних процесса. Реализовано фильтрационное правило для распределения строк по соответствующим каналам, а дочерние процессы корректно обрабатывают полученные строки, удаляя гласные и выводя результаты. Также были предусмотрены механизмы обработки системных ошибок, что повысило надежность работы программы. Таким образом, поставленные цели работы достигнуты, подтверждая освоение необходимых принципов и технологий.