Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №3 по курсу**

**«Операционные системы»**

Студент: Рамалданов Рустамхан Ражудинович

Группа: М8О-208Б-22

Вариант: 19

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2023

**Содержание**

1. Репозиторий

2. Постановка задачи

3. Общие сведения о программе

4. Общий метод и алгоритм решения

5. Исходный код

6. Демонстрация работы программы

7. Выводы

**Репозиторий**

<https://github.com/pepelulka/OS_labs>

**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 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.

**Общие сведения о программе**

Программа состоит из двух частей – главная программа, родительский процесс, описанный в main.c, и программа дочернего прочесса, описанная в child.c. Программы написаны под операционную систему Windows.

**Общий метод и алгоритм решения**

Родительский процесс получает на вход имена выходных файлов и создает дочерние процессы, передавая им имена файлов в качестве аргументов командной строки. Перед этим родительский процесс создает два shared-memory файла, по которым он будет передавать дочерним процессам строки, согласно правилу фильтрации. Дочерние процессы удаляют из строк гласные и записывают результат в соответствующие файлы.

**Исходный код**

**========================== parent.h ==========================**

**#pragma once**

**#ifdef \_WIN32**

**#include <iostream>**

**#include <windows.h>**

**namespace lab3 {**

**void ParentRoutine(const char\* pathToChild, std::istream &in);**

**}**

**#endif**

**========================== util.h ==========================**

**#pragma once**

**#ifdef \_WIN32**

**#include <windows.h>**

**#include <iostream>**

**#include <vector>**

**#include <cstdint>**

**#include <random>**

**namespace util {**

**// Writing to mmap for parent process**

**class TMMap {**

**private:**

**std::string pname, pmutexName;**

**size\_t psize, bufferSize;**

**HANDLE mmap;**

**// View starts with 4 bytes of header and then data**

**void \* view;**

**// If true - reading / else - writing**

**bool readingMode;**

**// Reading:**

**size\_t bytesRead = 0;**

**// Writing:**

**size\_t bytesWritten = 0;**

**public:**

**constexpr const static size\_t DEFAULT\_MAP\_SIZE = 1024;**

**constexpr const static size\_t HEADER\_SIZE = 4;**

**// mode == 'r' or mode == 'w'**

**TMMap(**

**char mode,**

**const std::string& mmapName,**

**size\_t mmapSize = DEFAULT\_MAP\_SIZE**

**);**

**TMMap() = delete;**

**TMMap(const TMMap&) = delete;**

**~TMMap();**

**uint32\_t GetBytesCount();**

**void Write(const std::string& line);**

**int Getc();**

**};**

**class Process {**

**private:**

**PROCESS\_INFORMATION info;**

**public:**

**Process() = delete;**

**Process(const Process&) = delete;**

**Process(const std::string &path, const std::vector<std::string> &args);**

**~Process();**

**void Terminate();**

**void Wait();**

**};**

**class Randomizer {**

**private:**

**std::mt19937 rng;**

**std::uniform\_int\_distribution<std::mt19937::result\_type> dist;**

**public:**

**Randomizer() {**

**std::random\_device dev;**

**rng = std::mt19937(dev());**

**dist = std::uniform\_int\_distribution<std::mt19937::result\_type>(1, 100);**

**}**

**bool Probability(size\_t percentage) {**

**return dist(rng) <= percentage;**

**}**

**};**

**bool IsVowel(char c);**

**}**

**#endif**

**========================== child.cpp ==========================**

**#include "util.h"**

**#include <iostream>**

**#include <fstream>**

**#include <cassert>**

**int main(int argc, char \*\*argv) {**

**if (argc < 3) {**

**std::cout << "Too few arguments\n";**

**exit(1);**

**}**

**std::ofstream out(argv[1]);**

**util::TMMap map('r', argv[2]);**

**for (size\_t i = 0; i < map.GetBytesCount(); i++) {**

**char c = map.Getc();**

**if (!util::IsVowel(c)) {**

**out << c;**

**}**

**}**

**out.close();**

**}**

**========================== parent.cpp ==========================**

**#include "../include/parent.h"**

**#include "../include/util.h"**

**#include <string>**

**#ifdef \_WIN32**

**namespace lab3 {**

**const std::string mmap1Name = "pepe\_mmap\_1";**

**const std::string mmap2Name = "pepe\_mmap\_2";**

**void ParentRoutine(const char\* pathToChild, std::istream &in) {**

**const size\_t firstProcessProbability = 80;**

**util::Randomizer rand;**

**std::string file1, file2;**

**std::getline(in, file1);**

**std::getline(in, file2);**

**util::TMMap map1('w', mmap1Name, 10000), map2('w', mmap2Name, 10000);**

**for (std::string line; std::getline(in, line);) {**

**// One extra empty string**

**if (rand.Probability(firstProcessProbability)) {**

**map1.Write(line + "\n");**

**} else {**

**map2.Write(line + "\n");**

**}**

**}**

**util::Process child1(pathToChild, {file1, mmap1Name});**

**util::Process child2(pathToChild, {file2, mmap2Name});**

**child1.Wait();**

**child2.Wait();**

**}**

**}**

**#endif**

**========================== util.cpp ==========================**

**#include "util.h"**

**#ifdef \_WIN32**

**namespace util {**

**TMMap::TMMap(**

**char mode,**

**const std::string& mmapName,**

**size\_t mmapSize**

**) : pname(mmapName), psize(mmapSize), bufferSize(psize + HEADER\_SIZE) {**

**if (mode == 'r') {**

**readingMode = true;**

**} else if (mode == 'w') {**

**readingMode = false;**

**} else {**

**throw std::invalid\_argument("Expected mode == 'r' or 'w'");**

**}**

**// Create mmap**

**mmap = CreateFileMappingA(**

**INVALID\_HANDLE\_VALUE,**

**nullptr,**

**PAGE\_READWRITE,**

**0,**

**bufferSize,**

**mmapName.c\_str()**

**);**

**if (mmap == nullptr) {**

**throw std::logic\_error("Can't create mmap");**

**}**

**view = MapViewOfFile(mmap, FILE\_MAP\_ALL\_ACCESS, 0, 0, bufferSize);**

**}**

**TMMap::~TMMap() {**

**UnmapViewOfFile(view);**

**CloseHandle(mmap);**

**}**

**uint32\_t TMMap::GetBytesCount() {**

**return \*((uint32\_t\*)view);**

**}**

**void TMMap::Write(const std::string& line) {**

**if (readingMode == true) {**

**throw std::logic\_error("Can't write in reading mode.");**

**}**

**if (bytesWritten + line.size() > psize) {**

**throw std::overflow\_error("Buffer overflow.");**

**}**

**uint32\_t\* uintView = (uint32\_t\*)view;**

**char \* bufferTop = (char\*)view + HEADER\_SIZE + bytesWritten;**

**// !!! Don't copy \0**

**memcpy(bufferTop, line.c\_str(), line.size());**

**bytesWritten += line.size();**

**\*uintView = bytesWritten;**

**}**

**int TMMap::Getc() {**

**if (readingMode == false) {**

**throw std::logic\_error("Can't read in writing mode.");**

**}**

**uint32\_t bytesCount = \*((uint32\_t\*)view);**

**if (bytesRead == bytesCount) {**

**return EOF;**

**}**

**char result = ((char\*)view)[HEADER\_SIZE + bytesRead];**

**bytesRead++;**

**return result;**

**}**

**Process::Process(const std::string &path, const std::vector<std::string> &args) {**

**STARTUPINFOA si;**

**ZeroMemory(&si, sizeof(si));**

**ZeroMemory(&info, sizeof(info));**

**std::string cmdArgs = path;**

**for (const auto& arg : args) {**

**cmdArgs += " " + arg;**

**}**

**char rawArgs[1024];**

**strcpy(rawArgs, cmdArgs.c\_str());**

**if(!CreateProcessA(path.c\_str(),**

**rawArgs,**

**nullptr,**

**nullptr,**

**TRUE,**

**0,**

**nullptr,**

**nullptr,**

**&si,**

**&info)) {**

**throw std::logic\_error("Can't create new process.");**

**return ;**

**}**

**}**

**void Process::Terminate() {**

**CloseHandle(info.hProcess);**

**CloseHandle(info.hThread);**

**}**

**Process::~Process() {**

**Terminate();**

**}**

**void Process::Wait() {**

**WaitForSingleObject(info.hProcess, INFINITE);**

**Terminate();**

**}**

**bool IsVowel(char c) {**

**return c == 'e' ||**

**c == 'u' ||**

**c == 'i' ||**

**c == 'o' ||**

**c == 'a';**

**}**

**}**

**#endif**

**========================== main.cpp ==========================**

**#include <iostream>**

**#include "include/parent.h"**

**using namespace lab3;**

**int main() {**

**char\* pathToChild;**

**if (getenv("PATH\_TO\_CHILD")) {**

**pathToChild = getenv("PATH\_TO\_CHILD");**

**} else {**

**std::cerr << "Error: PATH\_TO\_CHILD is not specified.\n";**

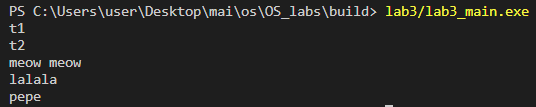
**exit(EXIT\_FAILURE);**

**}**

**ParentRoutine(pathToChild, std::cin);**

**}**

**Демонстрация работы программы**



**Выводы**

В ходе выполнения лабораторной работы я закрепил навыки работы с системными вызовами Windows. Выяснилось, что системный интерфейс Linux намного более дружелюбен к разработчику чем Windows.