

Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

**Лабораторная работа №1 по курсу
«Операционные системы»**

Студент: Тарасов Егор Дмитриевич
Группа: М8О-209Б-23
Вариант: 19
Преподаватель: Ядров Артём Леонидович
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2024

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

Постановка задачи

Цель работы

Приобретение практических навыков в:

1. Управление процессами в ОС
2. Обеспечение обмена данных между процессами посредством каналов

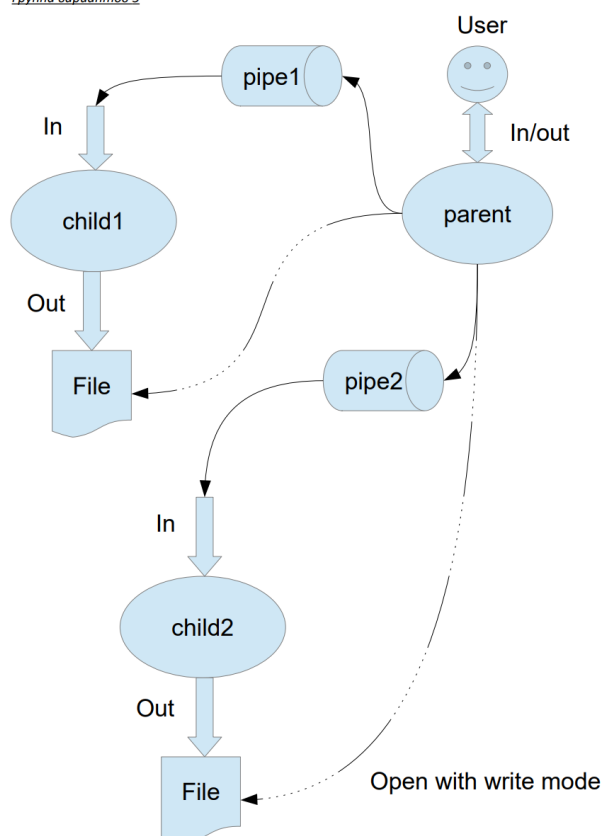
Задание

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 решения 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.

19 вариант) Правило фильтрации: с вероятностью 80% строки отправляются в pipe1, иначе в pipe2. Дочерние процессы удаляют все гласные из строк.

Группа вариантов 5



Общие сведения о программе

Программы собираются и компилируются при помощи CMake. В проекте существуют две программы: `parent.cpp` и `main.cpp`, предназначенные для описания родительского и дочерних процессов соответственно. В программе используются следующие функции Windows API:

1. **CreatePipe** – для создания каналов (аналог *pipe* в Unix)
2. **CreateProcess** – для создания дочерних процессов (аналог *fork* в Unix)
3. **WriteFile** – для записи данных в каналы (аналог *write* в Unix)
4. **ReadFile** – для чтения данных из каналов (аналог *read* в Unix)
5. **CloseHandle** – для закрытия дескрипторов (аналог *close* в Unix)
6. **WaitForSingleObject** – для ожидания завершения процесса (аналог *wait* в Unix)

Общий метод и алгоритм решения

Для реализации поставленной задачи необходимо:

- 1) Изучить принципы работы функций Windows API (`CreatePipe`, `CreateProcess`, `WriteFile`, `ReadFile`, `CloseHandle`, `WaitForSingleObject`)
- 2) Написать программу, которая будет работать с 3-мя процессами: один родительский и два дочерних, процессы связываются между собой при помощи `pipe`-ов.
- 3) Организовать работу по отбору используемого дочернего процесса для записи строк в файл, а также по непосредственно самой записи строк в файл.

Исходный код

parent.cpp

```
#include <windows.h>
#include <iostream>
#include <string>
#include <vector>
#include <cstdlib>
#include <ctime>

int main() {
    HANDLE hWritePipe1, hReadPipe1, hWritePipe2, hReadPipe2;
    SECURITY_ATTRIBUTES sa = {sizeof(SECURITY_ATTRIBUTES), NULL, TRUE};

    // Создаем каналы для общения с дочерними процессами
    if (!CreatePipe(&hReadPipe1, &hWritePipe1, &sa, 0) || !CreatePipe(&hReadPipe2,
&hWritePipe2, &sa, 0)) {
        std::cerr << "Failed to create pipes.\n";
        return 1;
    }

    std::string fileName1, fileName2;
    std::cout << "Enter the name of the file for child process 1: ";
    std::getline(std::cin, fileName1);
    std::cout << "Enter the name of the file for child process 2: ";
    std::getline(std::cin, fileName2);

    STARTUPINFO si1 = {sizeof(STARTUPINFO), 0}, si2 = {sizeof(STARTUPINFO), 0};
    PROCESS_INFORMATION pi1, pi2;
    ZeroMemory(&si1, sizeof(STARTUPINFO));
    ZeroMemory(&si2, sizeof(STARTUPINFO));
    si1.cb = sizeof(STARTUPINFO);
    si2.cb = sizeof(STARTUPINFO);
    si1.hStdError = si1.hStdOutput = hWritePipe1;
    si1.hStdInput = hReadPipe1;
    si1.dwFlags |= STARTF_USESTDHANDLES;
    si2.hStdError = si2.hStdOutput = hWritePipe2;
    si2.hStdInput = hReadPipe2;
    si2.dwFlags |= STARTF_USESTDHANDLES;

    char cmdLine1[256], cmdLine2[256];
    sprintf_s(cmdLine1, sizeof(cmdLine1), "ChildProcess1.exe %s",
fileName1.c_str());
    sprintf_s(cmdLine2, sizeof(cmdLine2), "ChildProcess2.exe %s",
fileName2.c_str());

    // Запуск дочерних процессов с именем файла в качестве аргумента
```

```

    if (!CreateProcess(NULL, cmdLine1, NULL, NULL, TRUE, 0, NULL, NULL, &si1,
&pi1)) {
        std::cerr << "Failed to create child process 1.\n";
        return 1;
    }
    if (!CreateProcess(NULL, cmdLine2, NULL, NULL, TRUE, 0, NULL, NULL, &si2,
&pi2)) {
        std::cerr << "Failed to create child process 2.\n";
        return 1;
    }

    // Заккрытие неиспользуемых концов каналов в родительском процессе
    CloseHandle(hReadPipe1);
    CloseHandle(hReadPipe2);

    std::srand(std::time(0)); // инициализация генератора случайных чисел
    std::string inputLine;

    // Чтение строки от пользователя и отправка ее в один из каналов
    while (std::getline(std::cin, inputLine)) {
        if (std::rand() % 100 < 80) { // 80% вероятность
            DWORD written;
            WriteFile(hWritePipe1, inputLine.c_str(), inputLine.length(), &written,
NULL);
        } else {
            DWORD written;
            WriteFile(hWritePipe2, inputLine.c_str(), inputLine.length(), &written,
NULL);
        }
    }

    // Заккрытие каналов
    CloseHandle(hWritePipe1);
    CloseHandle(hWritePipe2);

    // Ожидание завершения дочерних процессов
    WaitForSingleObject(pi1.hProcess, INFINITE);
    WaitForSingleObject(pi2.hProcess, INFINITE);

    CloseHandle(pi1.hProcess);
    CloseHandle(pi1.hThread);
    CloseHandle(pi2.hProcess);
    CloseHandle(pi2.hThread);

    return 0;
}

```

child.cpp

```
#include <windows.h>
#include <iostream>
#include <fstream>
#include <string>

std::string removeVowels(const std::string &input) {
    const std::string vowels = "aeiouAEIOU";
    std::string result;
    for (char ch : input) {
        if (vowels.find(ch) == std::string::npos) {
            result += ch;
        }
    }
    return result;
}

int main(int argc, char *argv[]) {
    if (argc < 2) {
        std::cerr << "File name argument is missing." << std::endl;
        return 1;
    }
    std::ofstream outFile(argv[1], std::ofstream::out);

    // Подготовка для чтения из стандартного ввода (канала)
    char buffer[256];
    DWORD bytesRead;

    while (true) {
        BOOL result = ReadFile(GetStdHandle(STD_INPUT_HANDLE), buffer, 255,
&bytesRead, NULL);
        if (!result || bytesRead == 0) break; // Выход из цикла, если чтение
завершилось
        buffer[bytesRead] = '\0'; // Обеспечение нуль-терминированной строки
        std::string output = removeVowels(std::string(buffer));
        if (outFile.is_open()) {
            outFile << output << std::endl;
        } else {
            std::cerr << "Failed to open file for writing." << std::endl;
            return 1;
        }
    }
    outFile.close();

    // Закрытие стандартного ввода
    CloseHandle(GetStdHandle(STD_INPUT_HANDLE));

    return 0;
}
```

Демонстрация работы программы

PS C:\Users\Xiaomi\Desktop\ll1\build> ./parent

Enter the name of the file for child process 1:

C:\Users\Xiaomi\Desktop\ll1\first.txt

Enter the name of the file for child process 2:

C:\Users\Xiaomi\Desktop\ll1\second.txt

Monkey D. Luffy

nami usop

SANJI Chopper

Nico Robin

FRRRRAAAAANKY!

Brook

PS C:\Users\Xiaomi\Desktop\ll1\build> cd ../

PS C:\Users\Xiaomi\Desktop\ll1> type first.txt

Mnky D. Lffy

zr

nm sp

SNJ Chppr

Nc Rbn

Brk

PS C:\Users\Xiaomi\Desktop\ll1> type second.txt

FRRRRNKY!

Выводы

Выполнив данную лабораторную работу, я понял принцип работы с процессами в ОС и обеспечения обмена данными между процессами посредством каналов. Я познакомился с понятием pipe, fork, узнал о существовании и принципах работы таких функций Windows API, обеспечивающих работу с процессами, как: CreatePipe, CreateProcess, WriteFile, ReadFile, CloseHandle и WaitForSingleObject. Кроме того, благодаря заданию моего варианта я узнал, как можно написать генератор случайных чисел на языке C++. Данная работа не только развила мои навыки работы с процессами ОС, но и улучшила моё понимание языка C++.