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: Тарасов Егор Дмитриевич
Группа: М8О-209Б-23
Вариант: 19
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2024

Содержание

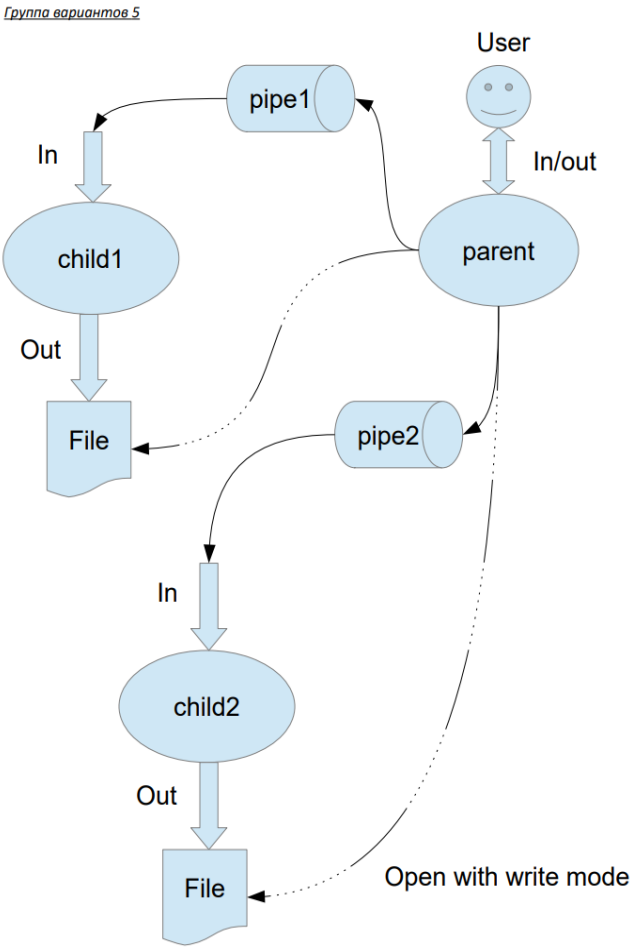
Репозиторий

https://github.com/EgorTarasov1/mai-os-labs

Постановка задачи

Цель работы
Приобретение практических навыков в:
· Освоение принципов работы с файловыми системами
· Обеспечение обмена данных между процессами посредством технологии «File mapping»

Задание
Составить и отладить программу на языке C++, 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.



19 вариант) с вероятностью 80% строки отправляются в pipe1, иначе в pipe2.

Дочерние процессы удаляют все гласные из строк.

Общие сведения о программе

Программа компилируется из файла parent.cpp. Также используются файлы child1.cpp и child2.cpp для реализации работы дочерних процессов. В программе используются следующие системные вызовы Windows API:

1. CreateFileMapping - создание отображаемого файла.
2. OpenFileMapping - открытие отображаемого файла в дочерних процессах..
3. MapViewOfFile — отображение памяти для чтения/записи.
4. UnmapViewOfFile - удаление отображения памяти (опционально).
5. CreateEvent - создание события для синхронизации.
6. OpenEvent - открытие события в дочерних процессах.
7. SetEvent - установка события в сигнальное состояние.
8. WaitForSingleObject - ожидание сигнала от события.
9. CreateProcess - создание дочерних процессов.
10. CloseHandle — освобождение ресурсов.
11. GetLastError — получение кода последней ошибки.

Общий метод и алгоритм решения

Для реализации поставленной задачи в среде Windows необходимо:

1. Изучить работу с отображаемыми файлами в память (функции CreateFileMapping, OpenFileMapping, MapViewOfFile и, при необходимости, UnmapView0fFile).
2. Изучить создание и управление процессами (функция CreateProcess).
3. Создать 2 дочерних и 1 родительский процесс.
4. В каждом процессе:
   - Открыть или создать объект отображаемого файла с использованием CreateFileMapping (В родительском процессе) и OpenFileMapping (в дочерних процессах).
   - Отобразить файл в память с использованием MapViewOfFile.
   - Выполнить преобразование данных в соответствии с вариантом задания.

Исходный код

Parent.cpp

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

#define MMAP_FILE_NAME "Local\\SharedMemory"
#define EVENT_NAME "Local\\DataReadyEvent"

int main() {
    HANDLE hFileMapping, hEvent;
    SECURITY_ATTRIBUTES sa = {sizeof(SECURITY_ATTRIBUTES), NULL, TRUE};

    hFileMapping = CreateFileMapping(INVALID_HANDLE_VALUE, &sa, PAGE_READWRITE,
0, 1024, MMAP_FILE_NAME);
    if (hFileMapping == NULL) {
        std::cerr << "Unable to create file mapping: " << GetLastError() <<
std::endl;
        return 1;
    }

    hEvent = CreateEvent(&sa, FALSE, FALSE, EVENT_NAME);
    if (hEvent == NULL) {
        std::cerr << "Unable to create event: " << GetLastError() << std::endl;
        CloseHandle(hFileMapping);
        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);

    // Дочерние процессы
    STARTUPINFO si1 = {sizeof(STARTUPINFO)};
    STARTUPINFO si2 = {sizeof(STARTUPINFO)};
    PROCESS_INFORMATION pi1, pi2;
    ZeroMemory(&pi1, sizeof(PROCESS_INFORMATION));
    ZeroMemory(&pi2, sizeof(PROCESS_INFORMATION));
```

```
    char cmdLine1[256], cmdLine2[256];
    sprintf_s(cmdLine1, sizeof(cmdLine1), "child1.exe \"%s\"", file-
Name1.c_str());
    sprintf_s(cmdLine2, sizeof(cmdLine2), "child2.exe \"%s\"", file-
Name2.c_str());

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

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

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

    LPVOID pBuffer = MapViewOfFile(hFileMapping, FILE_MAP_WRITE, 0, 0, 1024);
    if (pBuffer == NULL) {
        std::cerr << "Failed to map view of file.\n";
        CloseHandle(hFileMapping);
        CloseHandle(hEvent);
        return 1;
    }

    std::string inputLine;
    bool done = false;

    while (!done && std::getline(std::cin, inputLine)) {
        if (inputLine == "exit") {
            done = true;
            memcpy(pBuffer, "exit", 5);
            SetEvent(hEvent);
            break;
        }

        if (inputLine.size() >= 1024) {
            std::cerr << "Input line is too long, skipping.\n";
            continue;
        }
```

```
        memcpy(pBuffer, inputLine.c_str(), inputLine.size() + 1);
        std::cout << "Sending data: " << inputLine << std::endl;

        SetEvent(hEvent);

        std::this_thread::sleep_for(std::chrono::milliseconds(100));
    }

    WaitForSingleObject(pi1.hProcess, INFINITE);
    WaitForSingleObject(pi2.hProcess, INFINITE);

    CloseHandle(pi1.hProcess);
    CloseHandle(pi2.hProcess);
    CloseHandle(hFileMapping);
    CloseHandle(hEvent);

    return 0;
}
```

Child1.cpp/child2.cpp

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

#define MMAP_FILE_NAME "Local\\SharedMemory"
#define EVENT_NAME "Local\\DataReadyEvent"

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]);
    if (!outFile.is_open()) {
        std::cerr << "Failed to open file for writing." << std::endl;
        return 1;
```

```
    }

    HANDLE hFileMapping = OpenFileMapping(FILE_MAP_READ | FILE_MAP_WRITE, FALSE,
MMAP_FILE_NAME);
    if (hFileMapping == NULL) {
        std::cerr << "Failed to open file mapping: " << GetLastError() <<
std::endl;
        return 1;
    }

    HANDLE hEvent = OpenEvent(SYNCHRONIZE, FALSE, EVENT_NAME);
    if (hEvent == NULL) {
        std::cerr << "Failed to open event: " << GetLastError() << std::endl;
        CloseHandle(hFileMapping);
        return 1;
    }

    LPVOID pBuffer = MapViewOfFile(hFileMapping, FILE_MAP_READ, 0, 0, 1024);
    if (pBuffer == NULL) {
        std::cerr << "Failed to map view of file.\n";
        CloseHandle(hFileMapping);
        CloseHandle(hEvent);
        return 1;
    }

    while (true) {
        WaitForSingleObject(hEvent, INFINITE);

        std::string inputLine(static_cast<char*>(pBuffer));

        if (inputLine == "exit") {
            break;
        }

        std::cout << "Child received data: " << inputLine << std::endl;

        std::string output = removeVowels(inputLine);

        outFile << output << std::endl;
    }

    outFile.close();
    CloseHandle(hFileMapping);
    CloseHandle(hEvent);

    return 0;
}
```

Демонстрация работы программы

PS C:\Users\Xiaomi\Desktop\ll3\build> ./parent

Enter the name of the file for child process 1: C:\Users\Xiaomi\Desktop\ll3\first.txt

Enter the name of the file for child process 2: C:\Users\Xiaomi\Desktop\ll3\second.txt

abbacio

Sending data: abbacio

Child received data: abbacio

giorno

Sending data: giorno

PS C:\Users\Xiaomi\Desktop\ll3\build> ./parent

Enter the name of the file for child process 1: C:\Users\Xiaomi\Desktop\ll3\first.txt

Enter the name of the file for child process 2: C:\Users\Xiaomi\Desktop\ll3\second.txt

Giorno

Sending data: Giorno

Sending data: Bruno

Child received data: Bruno

Abbacio

Sending data: Abbacio

Child received data: Abbacio

Mista

Sending data: Mista

Child received data: Mista

Narancha

Sending data: Narancha

Child received data: Narancha

exit

First.txt:

Grn

Bbc

Nrnch

Second.txt:

Brn

Mst

Выводы

В C++ помимо механизма общения между процессами через pipe, также существуют и другие способы взаимодействия, например отображение файла в память, такой подход работает быстрее, за счет отсутствия постоянных вызовов read, write и тратит меньше памяти под кэш. После отображения возвращается void*, который можно привести к своему указателю на тип и обрабатывать данные как массив, где возвращенный указатель – указатель на первый элемент.