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: Леу Иван Александрович

Группа: М8О-209Б-23

Вариант: 9

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2024

**Содержание**

* Репозиторий
* Постановка задачи
* Общие сведения о программе
* Общий метод и алгоритм решения
* Исходный код
* Демонстрация работы программы
* Выводы

**Репозиторий**

[**https://github.com/IvanLeu/OS\_Labs/tree/main/lab3**](https://github.com/IvanLeu/OS_Labs/tree/main/lab3)

**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 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.

Вариант 9: В файле записаны команды вида:«число число число<endline>». Дочерний процесс

производит деление первого числа команда, на последующие числа в команде, а результат

выводит в стандартный поток вывода. Если происходит деление на 0, то тогда дочерний и

родительский процесс завершают свою работу. Проверка деления на 0 должна осуществляться на

стороне дочернего процесса. Числа имеют тип float. Количество чисел может быть произвольным.

**Общие сведения о программе**

Программа написана на языке С++ в операционной системе Windows. Для работы программы в командную строку следует ввести путь файла, с которого будут считаны данные.

**Общий метод и алгоритм решения**

Общая память состоит из самих данных и флага доступа, играющего роль синхронизатора доступа к памяти процессами. Родительский процесс записывает данные в общую память и устанавливает флаг готовности в памяти. Далее дочерний процесс, получивший доступ к данным, производит вычилсения, и результат перезаписывает в общую память. После этого устанавливается флаг в памяти, разрешающий родительскому процессу считать данные, и, после прочтения данных, родительский процесс выводит на экран результат. При деление на 0, все готовые результаты выводятся на экран и оба процесса завершаются.

**Исходный код**

main.cpp

#include <windows.h>

#include <stdexcept>

#include <iostream>

#include <string>

#include <cstring>

#define SHARED\_MEMORY\_NAME "Local\\MySharedMemory"

#define SHARED\_MEMORY\_SIZE 1024

struct SharedMemory {

char buffer[SHARED\_MEMORY\_SIZE];

volatile int flag; //0 - idle, 1 - data ready, 2 - result ready

};

int main() {

try {

std::string file\_name;

std::cin >> file\_name;

HANDLE hFile = CreateFileA(file\_name.c\_str(), GENERIC\_READ, 0, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (hFile == INVALID\_HANDLE\_VALUE) {

throw std::runtime\_error("Failed to open a file");

}

HANDLE hMapFile = CreateFileMappingA(

INVALID\_HANDLE\_VALUE, // Use the system paging file

NULL, // Default security

PAGE\_READWRITE, // Read/write access

0, // Maximum object size (high-order DWORD)

sizeof(SharedMemory), // Maximum object size (low-order DWORD)

SHARED\_MEMORY\_NAME // Name of the shared memory object

);

if (!hMapFile) {

throw std::runtime\_error("Failed to create file mapping");

}

SharedMemory\* lpBase = (SharedMemory\*)MapViewOfFile(

hMapFile, // Handle to map object

FILE\_MAP\_WRITE, // Read/write access

0, // File offset high

0, // File offset low

SHARED\_MEMORY\_SIZE // Number of bytes to map

);

if (!lpBase) {

CloseHandle(hMapFile);

throw std::runtime\_error("Failed to map view of file");

}

lpBase->flag = 0;

TCHAR szCmdLine[] = TEXT("child.exe");

PROCESS\_INFORMATION piProcInfo;

STARTUPINFO siStartInfo;

ZeroMemory(&piProcInfo, sizeof(PROCESS\_INFORMATION));

ZeroMemory(&siStartInfo, sizeof(STARTUPINFO));

siStartInfo.cb = sizeof(STARTUPINFO);

if (!CreateProcess(

NULL,

szCmdLine,

NULL,

NULL,

FALSE,

0,

NULL,

NULL,

&siStartInfo,

&piProcInfo)) {

throw std::runtime\_error("Failed to create a child process");

}

char buffer[SHARED\_MEMORY\_SIZE];

DWORD bytesRead;

while (ReadFile(hFile, buffer, sizeof(buffer), &bytesRead, NULL) && bytesRead > 0) {

memcpy(lpBase->buffer, buffer, bytesRead);

lpBase->flag = 1;

do {}

while (lpBase->flag == 1);

if (lpBase->flag == 2) {

std::cout << "Result: " << static\_cast<char\*>(lpBase->buffer) << std::endl;

memset(lpBase->buffer, 0, SHARED\_MEMORY\_SIZE);

lpBase->flag = 0;

}

}

CloseHandle(hFile);

WaitForSingleObject(piProcInfo.hProcess, INFINITE);

UnmapViewOfFile(lpBase);

CloseHandle(hMapFile);

CloseHandle(piProcInfo.hProcess);

CloseHandle(piProcInfo.hThread);

}

catch (const std::runtime\_error& e) {

std::cerr << "Error: " << e.what() << std::endl;

}

catch (...) {

std::cerr << "Unknown error occurred" << std::endl;

}

return 0;

}

child.cpp

#include <windows.h>

#include <string>

#include <sstream>

#include <iostream>

#include <cstring>

#define SHARED\_MEMORY\_NAME "Local\\MySharedMemory"

#define SHARED\_MEMORY\_SIZE 1024

struct SharedMemory {

char buffer[SHARED\_MEMORY\_SIZE];

volatile int flag; //0 - idle, 1 - data ready, 2 - result ready

};

int main() {

try {

HANDLE hMapFile = OpenFileMappingA(

FILE\_MAP\_WRITE, // Read/write access

FALSE, // Do not inherit the handle

SHARED\_MEMORY\_NAME

);

if (!hMapFile) {

throw std::runtime\_error("Child: Failed to open file mapping");

}

SharedMemory\* lpBase = (SharedMemory\*)MapViewOfFile(

hMapFile, // Handle to map object

FILE\_MAP\_WRITE, // Read/write access

0, // File offset high

0, // File offset low

sizeof(SharedMemory)

);

if (!lpBase) {

CloseHandle(hMapFile);

throw std::runtime\_error("Child: Failed to map view of file");

}

while (true) {

if (lpBase->flag == 1) {

char buffer[SHARED\_MEMORY\_SIZE];

memcpy(buffer, lpBase->buffer, SHARED\_MEMORY\_SIZE);

std::string input(buffer);

if (input.empty()) {

break;

}

std::istringstream iss(input);

std::string line;

std::ostringstream resultBuffer;

while (std::getline(iss, line)) {

std::istringstream line\_ss(line);

float result;

line\_ss >> result;

float y = 1.0f;

while (line\_ss >> y) {

if (y == 0.0f) {

std::string resultStr = resultBuffer.str();

strncpy(static\_cast<char\*>(lpBase->buffer), resultStr.c\_str(), SHARED\_MEMORY\_SIZE);

lpBase->flag = 2;

throw std::runtime\_error("Child: Division by zero");

}

result /= y;

}

resultBuffer << result << " ";

}

std::string resultStr = resultBuffer.str();

strncpy(static\_cast<char\*>(lpBase->buffer), resultStr.c\_str(), SHARED\_MEMORY\_SIZE);

lpBase->flag = 2;

}

else if (lpBase->flag == 0) {

break;

}

}

UnmapViewOfFile(lpBase);

CloseHandle(hMapFile);

}

catch (const std::runtime\_error& e) {

std::cerr << e.what() << std::endl;

return 1;

}

catch (...) {

std::cerr << "Unknown error occurred" << std::endl;

return 1;

}

return 0;

}

**Демонстрация работы программы**

>>file.txt

>>Result: 1 0 13.3333 4

**Выводы**

Memory-mapped files — это механизм, позволяющий отобразить содержимое файла напрямую в адресное пространство процесса. Это позволяет работать с файлами как с обычной оперативной памятью, без необходимости явного чтения и записи данных, что существенно упрощает и ускоряет работу с большими файлами. Такой подход особенно удобен для обработки больших объемов данных, совместного доступа нескольких процессов к одному файлу, а также для реализации эффективных кешей. Memory-mapped files — это мощный инструмент, который объединяет удобство использования и высокую производительность.

Memory-mapped files в C++ можно использовать с помощью функций операционной системы. На Windows это функции CreateFileMapping и MapViewOfFile из библиотеки Windows.h, а на POSIX-системах — функция mmap из библиотеки sys/mman.h. Эти инструменты обеспечивают низкоуровневый доступ к функционалу memory-mapped files и позволяют эффективно работать с файлами на уровне операционной системы.