МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования «Южно-Уральский государственный университет»

(национальный исследовательский университет)

Высшая школа электроники и компьютерных наук

Кафедра системного программирования

ОТЧЕТ

по практической работе № 2

по дисциплине   
«Теория, методы и средства параллельной обработки информации»

Выполнил:

студент группы КЭ–301

/ Д.А. Иванов /

(подпись)

« » 2024 г.

**Задача:**

Создать параллельное приложение, состоящее из 2 процессов, выполняющих обмен информацией через файл, отображаемый в память (файл на диске не создавать!): 1й процесс посылает 2му ОДНО сообщение, 2й процесс его получает и выводит на экран. Для работы с отображаемым в память файлом использовать функции WinAPI (или другой ОС).

**Листинг программы:**

#include <iostream>

#include <windows.h>

#include <string>

#include <thread>

int process\_1()

{

HANDLE hMapFile = CreateFileMapping(INVALID\_HANDLE\_VALUE, nullptr, PAGE\_READWRITE, 0, 256,

"SharedMemory");

if (hMapFile == nullptr) {

std::cerr << "Failed to create shared memory." << std::endl;

return 1;

}

LPVOID Buf = MapViewOfFile(hMapFile, FILE\_MAP\_WRITE, 0, 0, 256);

if (Buf == nullptr) {

std::cerr << "Failed to map view of file." << std::endl;

CloseHandle(hMapFile);

return 1;

}

HANDLE semaphore\_1 = CreateSemaphore(nullptr, 0, 1, "Semaphore\_1");

HANDLE semaphore\_2 = CreateSemaphore(nullptr, 0, 1, "Semaphore\_2");

std::cout << "Enter a message: ";

std::string message;

std::getline(std::cin, message);

CopyMemory(Buf, message.c\_str(), message.size() + 1);

ReleaseSemaphore(semaphore\_1, 1, nullptr);

WaitForSingleObject(semaphore\_2, INFINITE);

UnmapViewOfFile(Buf);

CloseHandle(semaphore\_1);

CloseHandle(semaphore\_2);

CloseHandle(hMapFile);

return 0;

}

int process\_2()

{

HANDLE hMapFile = OpenFileMapping(FILE\_MAP\_READ, FALSE, "SharedMemory");

if (hMapFile == nullptr) {

std::cerr << "Failed to open shared memory." << std::endl;

return 1;

}

LPVOID Buf = MapViewOfFile(hMapFile, FILE\_MAP\_READ, 0, 0, 256);

if (Buf == nullptr) {

std::cerr << "Failed to map view of file." << std::endl;

CloseHandle(hMapFile);

return 1;

}

HANDLE semaphore\_1 = OpenSemaphore(SEMAPHORE\_ALL\_ACCESS, FALSE, "Semaphore\_1");

HANDLE semaphore\_2 = OpenSemaphore(SEMAPHORE\_ALL\_ACCESS, FALSE, "Semaphore\_2");

WaitForSingleObject(semaphore\_1, INFINITE);

std::cout << "Received message: " << (char\*)Buf << std::endl;

ReleaseSemaphore(semaphore\_2, 1, nullptr);

UnmapViewOfFile(Buf);

CloseHandle(semaphore\_1);

CloseHandle(semaphore\_2);

CloseHandle(hMapFile);

return 0;

}

int main() {

std::thread t1(process\_1);

std::thread t2(process\_2);

t1.join();

t2.join();

return 0;

}

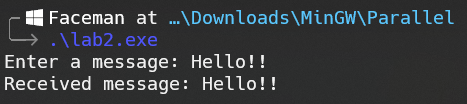


Рисунок 1 – Результат выполнения программы

**Ответы на вопросы:**

1) Привести определение файла, отображаемого в память, пояснить причину появления этого механизма и способы его использования:

Файл, отображаемый в память – это файл на диске, который был скопирован в виртуальную память процесса и теперь доступен в виде области памяти для чтения и записи. Это делает файл доступным для обработки обычных данных в памяти, с помощью привычных операций с указателями. Появление механизма отображения файлов в память обусловлено несколькими причинами. Одной из главных является улучшение производительности программы за счет того, что чтение и запись данных в файл осуществляется прямо через память, минуя стандартные операции ввода/вывода. Это позволяет избежать множественных операций чтения/записи на диск, что может быть заметно быстрее и эффективнее. Способы использования файлов, отображаемых в память, могут быть разнообразны. Например, это может быть использовано для быстрой загрузки больших объемов данных в оперативную память, для обработки данных в формате файла, без необходимости читать и записывать их по частям. Также это может быть использовано для совместной работы нескольких процессов с одним файлом, обеспечивая им общий доступ к данным без необходимости копирования данных между процессами.

2) Как реализуется синхронизация операций чтения/записи и почему?

Для синхронизации операций чтения/записи используются семафоры semaphore\_1 и semaphore\_2. Семафор semaphore\_1 используется отправителем для сообщения получателю о том, что данные готовы к чтению. Когда отправитель помещает данные в общую память, он освобождает semaphore\_1, а получатель ожидает, пока семафор не будет освобожден. Затем, когда получатель завершает чтение данных, он освобождает семафор semaphore\_2, чтобы отправитель мог продолжить выполнение. Таким образом, семафоры гарантируют, что чтение и запись данных происходят синхронно и без конфликтов.