Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет

информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные среды и системное программирование

ОТЧЁТ

к лабораторной работе №4

на тему

**ВЗАИМОДЕЙСТВИЕ ПРОЦЕССОВ(ПОТОКОВ):**

**ВЗАИМНОЕ ИСКЛЮЧЕНИЕ И СИНХРОНИЗАЦИЯ**

|  |
| --- |
| Выполнил: студент гр. 253503  Котова К.А. |
| Проверил: ассистент кафедры информатики Гриценко Н.Ю. |

Минск 2024

**СОДЕРЖАНИЕ**

[1 Формулировка задачи 3](#_Toc182434332)

[2 Описание основных функций программы 4](#_Toc182434333)

[2.1 Функция создания и отображения файла 4](#_Toc182434334)

[2.2 Функция записи в файл 4](#_Toc182434335)

[2.3 Функция чтения записей из файла 4](#_Toc182434336)

[3 Результаты выполнения программы 5](#_Toc182434337)

[3.1 Равное количество писателей и читателей 5](#_Toc182434338)

[3.2 Различное количество писателей и читателей 5](#_Toc182434339)

[Вывод 7](#_Toc182434340)

[Список использованных источников 8](#_Toc182434341)

[Приложение А (справочное) 9](#_Toc182434342)

# 

# 1 ФОРМУЛИРОВКА ЗАДАЧИ

Организация доступа к содержимому разделяемой памяти для параллельно выполняющихся процессов (потоков) с предотвращением коллизий. Близость к модели «писатели-читатели».

Эксперименты (тесты) с целью определения количественных характеристик – прогоны с разными параметрами системы.

Примеры практического использования: файл (банк/база данных), разделяемый несколькими процессами (близко к модели «писатели-читатели»); высокопроизводительный буфер «контейнеров» с данными для использования множеством потребителей без перераспределения памяти для «контейнеров и без копирования их содержимого.

Пример подхода к реализации: фрагментация памяти на блоки фиксированного (или переменного) размера, предоставляемые потребителям (процессам, потокам); занятый блок временно блокируется для доступа со стороны других потребителей. Массив (список) блоков, массив (список) семафоров (мьютексов), управляющих доступом к блокам. API для запроса, освобождения и др. действий с блоками.

Изменяемые параметры модели: общий объем разделяемой памяти; размер (количество) фрагментов; количество потребителей; характеристика запросов к памяти и др.

# 2 ОПИСАНИЕ ОСНОВНЫХ ФУНКЦИЙ ПРОГРАММЫ

## 2.1 Функция создания и отображения файла

CreateAndMapFile – это функция, которая выполняет создание файла и его отображение в память. Сначала она вызывает функцию CreateFile, чтобы создать или открыть файл с правами на чтение и запись. Если создание файла завершилось неудачно, функция выводит сообщение об ошибке и завершает выполнение. Далее она использует функции SetFilePointer [1] и SetEndOfFile [2], чтобы установить начальный размер файла. Это нужно для того, чтобы файл был готов к использованию с определенным размером. После этого, с помощью функции CreateFileMapping [3], создается отображение файла в память.

Если операция создания отображения завершилась успешно, функция вызывает MapViewOfFile [4], которая отображает файл в виртуальной памяти. Это позволяет программе работать с файлом как с обычной памятью. Если на любом этапе происходит ошибка, выводится сообщение, и файл не создается.

## 2.2 Функция записи в файл

Функция WriteRecord реализована как потоковая функция, которая выполняет запись записи в отображенный файл. Она принимает структуру WriteParams как параметр, содержащий адрес отображенной памяти, индекс записи, которую нужно изменить, идентификатор записи и имя. Сначала она блокирует доступ к общим данным, вызывая WaitForSingleObject [5] на мьютексе hSharedMutex. Далее она получает указатель на нужную запись в отображенной памяти, используя переданный индекс, и заполняет его полями id и name. После записи данных она снимает блокировку с hSharedMutex и выводит результат в консоль. Вывод в консоль также защищен мьютексом hCoutMutex, чтобы избежать перекрытия сообщений от разных потоков.

## 2.3 Функция чтения записей из файла

Функция ReadRecord, также реализованная как потоковая, выполняет чтение записи из отображенного файла. Как и WriteRecord, она принимает параметр в виде структуры ReadParams, содержащей адрес отображенной памяти и индекс нужной записи. Для обеспечения согласованности данных, функция сначала блокирует доступ к общей памяти с помощью WaitForSingleObject на мьютексе hSharedMutex, затем извлекает данные записи по индексу и выводит их в консоль. Вывод, как и в функции записи, защищен мьютексом hCoutMutex.

# 3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ПРОГРАММЫ

## 3.1 Равное количество писателей и читателей

При равном количестве потоков-писателей и потоков-читателей, например, по пять, программа работает следующим образом: сначала запускается пять потоков, каждый из которых записывает запись с уникальным идентификатором и именем Writer в определённые индексы массива записей. Затем пять потоков-читалелей обращаются к этим же индексам и выводят данные, которые ранее записали потоки-писатели.

Результат работы: каждый писатель сообщает в консоль о своей записи, а каждый читатель затем выводит на консоль считанную запись. При правильной синхронизации результат должен выглядеть так: сообщения о записи данных от потоков-писателей и далее сообщения от потоков-читателей с идентичными значениями.

Результат работы программы представлен на рисунке 3.1.

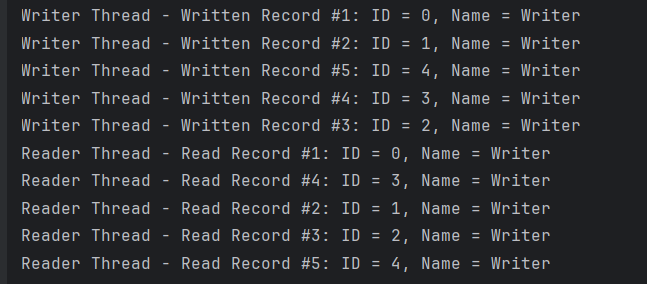


Рисунок 3.1 – Pезультат работы программы

## 3.2 Различное количество писателей и читателей

Результат программы на рисунке 3.2 показывает, как потоки-писатели и потоки-читатели работают параллельно, что приводит к интересным эффектам из-за разного количества потоков записи и чтения.

В начале мы видим, что первые потоки-писатели последовательно записывают записи с ID от 0 до 5, заполняя первые несколько индексов массива. Однако, поскольку писателей больше, чем читателей, часть записей перезаписывается новыми потоками-писателями до того, как их успевают прочитать потоки-читатели.

Читатели читают записи в момент их запуска, что иногда приводит к тому, что они видят уже перезаписанные данные.

Таким образом, мы видим, что при большем количестве писателей часть данных перезаписывается до того, как их успевают прочитать, а некоторые записи читаются с обновленными значениями, так как читатели получают данные в момент вызова, а не в момент их первой записи.

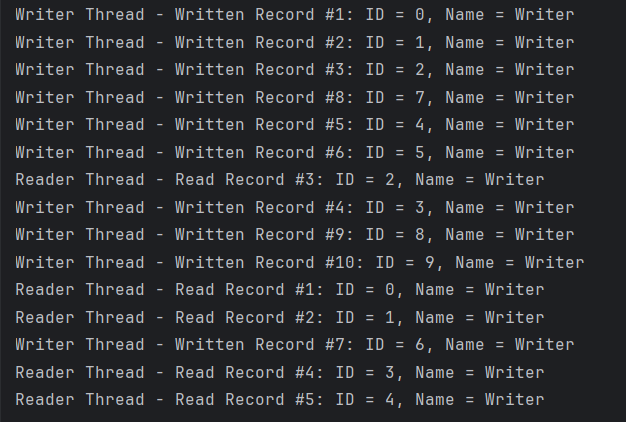


Рисунок 3.2 – Результат работы программы

# ВЫВОД

В ходе выполнения данной лабораторной работы была реализована модель организации доступа к разделяемой памяти для параллельно работающих потоков с использованием модели «писатели-читатели». В качестве основы использовался API Windows, позволяющий создавать отображение файла в память и работать с ним как с разделяемым массивом данных. Модель «писатели-читатели» обеспечивает параллельное чтение несколькими потоками, при этом запись блокирует доступ для других потоков. Этот подход важен для систем, где необходимо поддерживать целостность данных при параллельной обработке, таких как базы данных и кеширующие системы.

Создание файла и отображение его в память позволили организовать работу с ним как с массивом структур данных, где каждый поток может обращаться к отдельному блоку памяти, упрощая доступ и управление данными. Такой подход полезен в приложениях, требующих частой модификации данных, разделяемых между процессами или потоками. Применение мьютекса дало возможность параллельного чтения, повышая производительность в случаях, когда операции чтения происходят чаще записей.

Таким образом, выполнение данной лабораторной работы позволило на практике освоить методы организации доступа к разделяемой памяти в многопоточной среде и углубить понимание модели «писатели-читатели». Реализованный подход может применяться в системах, требующих высокой скорости чтения данных и возможности параллельной работы с памятью.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Документация Microsoft для функции SetFilePointer [Электронный ресурс]. – Электронные данные. – Режим доступа https://learn.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-

[2] Документация Microsoft для функции SetEndOfFile [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://learn.microsoft.com/> de-de/windows/win32/api/fileapi/nf-fileapi-setendoffile –Дата доступа: 05.10.2024.

[3] Документация Microsoft для функции CreateFileMapping [Электронный ресурс]. – Электронные данные. – Режим доступа: https:// learn.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-createfilemappinga – Дата доступа: 05.10.2024.

[4] Документация Microsoft для функции MapViewOfFile [Электронный ресурс]. – Электронные данные. – Режим доступа: https:// learn.microsoft.com/en-us/windows/win32/api/memoryapi/nf-memoryapi-mapviewoffile –Дата доступа: 05.10.2024.

[5] Документация Microsoft для функции WaitForSingleObject – Режим доступа: https://learn.microsoft.com/en-us/ windows/win32/api/synchapi/ nf-synchapi-waitforsingleobject – Дата доступа: 05.10.2024.

[6] Документация Microsoft SRW-locks – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/sync/slim-reader-writer--srw--locks> – Дата доступа: 05.10.2024.

# ПРИЛОЖЕНИЕ А (справочное) Код программы

Содержимое файла main.cpp

#include <windows.h>

#include <iostream>

#include <string>

#include <vector>

struct Record {

int id;

char name[50];

};

std::vector<bool> dataReady;

SRWLOCK srwLock;

HANDLE hCoutMutex;

struct WriteParams {

LPVOID mappedView;

int index;

int id;

const char\* name;

};

struct ReadParams {

LPVOID mappedView;

int index;

};

HANDLE CreateAndMapFile(LPCSTR filename, DWORD size, HANDLE &hFile, LPVOID &mappedView) {

hFile = CreateFile(filename, GENERIC\_READ | GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

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

std::cerr << "Failed to create file" << std::endl;

return NULL;

}

if (SetFilePointer(hFile, size, NULL, FILE\_BEGIN) == INVALID\_SET\_FILE\_POINTER && GetLastError() != NO\_ERROR) {

std::cerr << "Failed to set file pointer with error: " << GetLastError() << std::endl;

CloseHandle(hFile);

return NULL;

}

if (!SetEndOfFile(hFile)) {

std::cerr << "Failed to set end of file with error: " << GetLastError() << std::endl;

CloseHandle(hFile);

return NULL;

}

HANDLE hMapFile = CreateFileMapping(hFile, NULL, PAGE\_READWRITE, 0, size, NULL);

if (!hMapFile) {

std::cerr << "Failed to create file mapping" << std::endl;

CloseHandle(hFile);

return NULL;

}

mappedView = MapViewOfFile(hMapFile, FILE\_MAP\_ALL\_ACCESS, 0, 0, size);

if (!mappedView) {

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

CloseHandle(hMapFile);

CloseHandle(hFile);

return NULL;

}

return hMapFile;

}

DWORD WINAPI WriteRecord(LPVOID params) {

auto\* wp = static\_cast<WriteParams\*>(params);

AcquireSRWLockExclusive(&srwLock); // Эксклюзивный доступ для записи

Record\* records = static\_cast<Record\*>(wp->mappedView);

records[wp->index].id = wp->id;

strncpy(records[wp->index].name, wp->name, sizeof(records[wp->index].name) - 1);

records[wp->index].name[sizeof(records[wp->index].name) - 1] = '\0';

dataReady[wp->index] = true; // Устанавливаем флаг готовности

ReleaseSRWLockExclusive(&srwLock); // Освобождаем эксклюзивный доступ

WaitForSingleObject(hCoutMutex, INFINITE);

std::cout << "Writer Thread - Written Record #" << wp->index + 1 << ": ID = " << wp->id << ", Name = " << wp->name << std::endl;

ReleaseMutex(hCoutMutex);

delete wp;

return 0;

}

DWORD WINAPI ReadRecord(LPVOID params) {

auto\* rp = static\_cast<ReadParams\*>(params);

if (dataReady[rp->index]) {

AcquireSRWLockShared(&srwLock); // Совместный доступ для чтения

Record\* records = static\_cast<Record\*>(rp->mappedView);

WaitForSingleObject(hCoutMutex, INFINITE);

std::cout << "Reader Thread - Read Record #" << rp->index + 1 << ": ID = " << records[rp->index].id << ", Name = " << records[rp->index].name << std::endl;

ReleaseMutex(hCoutMutex);

ReleaseSRWLockShared(&srwLock); // Освобождаем доступ для чтения

} else {

WaitForSingleObject(hCoutMutex, INFINITE);

std::cout << "Reader Thread - Record #" << rp->index + 1 << " is not yet written, skipping." << std::endl;

ReleaseMutex(hCoutMutex);

}

delete rp;

return 0;

}

int main() {

LPVOID mappedView;

HANDLE hFile;

int maxRecords = 10;

const DWORD initialSize = sizeof(Record) \* maxRecords;

InitializeSRWLock(&srwLock);

hCoutMutex = CreateMutex(NULL, FALSE, NULL);

HANDLE hMapFile = CreateAndMapFile("database2.bin", initialSize, hFile, mappedView);

if (hMapFile) {

dataReady.resize(maxRecords, false);

HANDLE threads[10];

for (int i = 0; i < 5; i++) {

auto\* params = new WriteParams{mappedView, i, i, "Writer"};

threads[i] = CreateThread(NULL, 0, WriteRecord, params, 0, NULL);

}

for (int i = 0; i < 5; i++) {

auto\* params = new ReadParams{mappedView, i};

threads[i + 5] = CreateThread(NULL, 0, ReadRecord, params, 0, NULL);

}

WaitForMultipleObjects(10, threads, TRUE, INFINITE);

for (auto& thread : threads) {

CloseHandle(thread);

}

UnmapViewOfFile(mappedView);

CloseHandle(hMapFile);

CloseHandle(hFile);

}

CloseHandle(hCoutMutex);

return 0;

}