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.253504 Сапроненко В.В.

Проверил: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2024

**СОДЕРЖАНИЕ**

[1 Формулировка задачи 3](#_30j0zll)

[2 Краткие теоретические сведения 4](#_1fob9te)

[3 Описание функций программы 6](#_3znysh7)

[3.1 Структура данных для раздельного доступа 6](#_2et92p0)

[3.2 Создание набора мьютексов 6](#_tyjcwt)

[3.3 Организация работы с разделяемой памятью 6](#_tyjcwt)

[3.4 Рассылка сообщения всем клиентам 7](#_7g1mj7cc96qc)

З[аключение 8](#_6hzpozfdz8ye)

С[писок использованных источников 9](#_1t3h5sf)

П[риложение](#_4d34og8) А [(обязательное)](#_bdnxrcn5v3lk) И[сходный код программы 10](#_2e6ri3n970em)

# 1 ФОРМУЛИРОВКА ЗАДАЧИ

Цель этой задачи заключается в создании и исследовании механизма синхронизации для параллельно работающих процессов или потоков, обеспечивающего безопасный доступ к общей памяти. Это должно предотвратить коллизии и учитывать особенности модели «читатели-писатели».

Ключевые функциональные требования включают разработку системы, которая обеспечивает взаимное исключение с помощью мьютексов или семафоров, а также синхронизацию доступа для предотвращения конфликтов между писателями и читателями. Важным этапом является проведение тестирования с различными системными параметрами.

Изменяемыми параметрами модели могут быть общий объем разделяемой памяти, размеры и количество блоков памяти, число процессов (читателей и писателей), а также характеристики запросов, такие как частота операций чтения и записи.

Для реализации предполагается разделение памяти на блоки фиксированного или переменного размера с использованием управляющих структур. Массив блоков памяти определяет доступные сегменты, а массив семафоров или мьютексов управляет доступом к каждому блоку. Необходимо разработать API, который позволит выполнять такие действия, как запрос блоков для чтения или записи, их освобождение и мониторинг состояния памяти.

Технические эксперименты предполагают проведение серии запусков системы с разными параметрами, например, с различным количеством читателей и писателей, изменением размеров блоков и увеличением общей нагрузки. Измеряются такие характеристики, как время выполнения операций чтения и записи, пропускная способность системы и задержки при ожидании доступа.

В качестве практической задачи необходимо разработать программу для демонстрации разделения памяти, используя каналы Windows. Задача включает следующие этапы:

– создание структур данных для раздельного доступа;

– разработка набора механизмов для контроля доступа к данным;

– создание алгоритма организации работы с данными;

– тестирование программы с использованием множества потоков.

В результат е выполнения этой лабораторной работы будут приобретены не только теоретические знания работы со средствами межпроцессного взаимодействия (IPC), но и практический опыт в разработке приложений, эффективно использующих ресурсы компьютера.

# 2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Взаимное исключение и синхронизация являются важными задачами при взаимодействии процессов и потоков. Эти механизмы управляют доступом к общим ресурсам, предотвращая конфликты и некорректное поведение программ. Взаимное исключение обеспечивает доступ к критическому ресурсу только одному процессу или потоку в каждый момент времени, что исключает ситуации одновременного изменения данных несколькими процессами. Синхронизация гарантирует упорядоченность действий между потоками, позволяя, например, одному потоку читать данные только после их записи другим потоком.

Критический ресурс — это объект или данные, которые одновременно могут использоваться только одним процессом или потоком. Примеры критических ресурсов включают разделяемую память, файлы или оборудование, такое как принтеры. Критическая секция — это участок кода, который обращается к критическому ресурсу. Для предотвращения конфликтов выполнение кода в критической секции ограничивается одним потоком или процессом[1].

Проблема атомарности возникает, когда операции проверки и модификации данных выполняются отдельно, что позволяет другому потоку вмешаться между этими операциями и нарушить консистентность данных. Например, если два процесса одновременно проверяют значение переменной и изменяют ее, результат может быть некорректным. Атомарность гарантирует, что операция либо выполняется полностью, либо не выполняется вовсе, и достигается с использованием специальных механизмов, таких как инструкции блокировки на уровне процессора или синхронизационные объекты.

Для анализа ситуаций синхронизации применяются различные теоретические модели. В модели «писатели-читатели» процессы делятся на группы: читатели получают доступ только на чтение, а писатели изменяют данные. Задача заключается в том, чтобы несколько читателей могли работать одновременно, но писатели не могли вмешиваться во время чтения или записи. Модель «производитель-потребитель» описывает взаимодействие, при котором один процесс генерирует данные, а другой их обрабатывает, что требует предотвращения переполнения или опустошения буфера. Семафорная модель использует счетчики для управления доступом к ресурсам с ограниченной пропускной способностью, например, к фиксированному числу подключений. Модель барьеров применяется в ситуациях, когда выполнение группы потоков должно синхронизироваться на определённом этапе программы.

Для решения задач синхронизации используются различные механизмы. Critical Section позволяет ограничить доступ к коду внутри одного процесса, блокируя другие потоки, которые пытаются выполнить этот код. Объекты ожидания, такие как мьютексы, семафоры, события и таймеры, предоставляют инструменты для управления доступом и синхронизации. Мьютекс обеспечивает взаимное исключение, разрешая доступ только одному процессу или потоку. Семафор управляет доступом с помощью счётчика, блокируя потоки при отсутствии свободных ресурсов. События позволяют синхронизировать потоки, уведомляя их о наступлении определённых условий, а ожидаемый таймер задает временные интервалы для выполнения операций. Функции ожидания, такие как WaitForSingleObject и WaitForMultipleObjects, используются для блокировки потока до освобождения объекта или достижения определённых условий[2].

Барьеры используются для синхронизации группы потоков, блокируя их до тех пор, пока все потоки не достигнут определенного состояния. Спин-блокировки представляют собой активный способ ожидания, при котором поток в цикле проверяет доступность ресурса, не переходя в состояние блокировки. Этот метод применим в случаях, когда ожидается короткое время ожидания, чтобы избежать накладных расходов на переключение контекста[3].

Эти механизмы и модели обеспечивают корректное выполнение параллельных программ и решают широкий спектр задач, связанных с синхронизацией и управлением доступом к ресурсам.

# 

# 3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Согласно формулировке задачи, были спроектированы следующие функции программы :

– создание структуры данных для раздельного доступа;

– создание набора сущностей для контроля доступа к данным;

– создание алгоритма организации работы с данными;

## 3.1 Структура данных для раздельного доступа

Структура разделяемой памяти используются две основные структуры (рисунок 3.1).

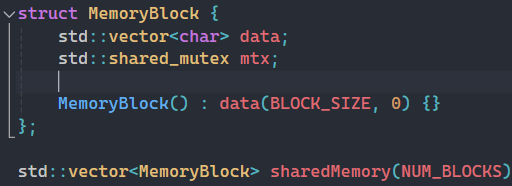


Рисунок 3.1 – Структура разделяемой памяти

## 3.2 Создание набора мьютексов

На каждый блок разделяемой памяти создается объект мьютекса (рисунок 3.2).

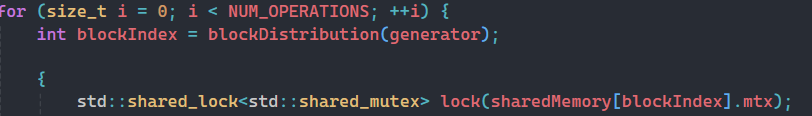


Рисунок 3.2 – Создание мьютексов

## 3.3 Организация работы с разделяемой памятью

В коде реализована модель «читатели-писатели» для организации безопасного доступа к разделяемой памяти несколькими потоками. Разделяемая память представлена в виде массива блоков, к которым потоки могут обращаться для чтения или записи. Ниже представлены функции для работы с блоками(рисунок 3.3).

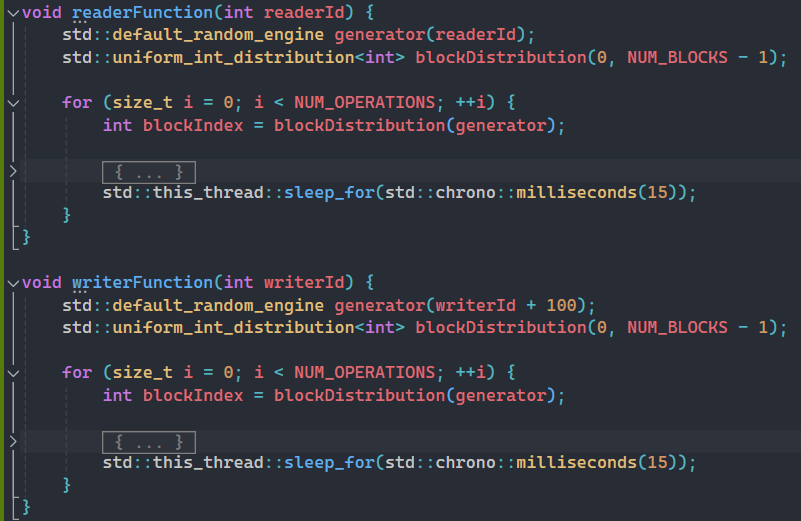


Рисунок 3.3 – Функции работы с блоками

Взаимное исключение и синхронизация являются фундаментальными аспектами взаимодействия процессов и потоков, обеспечивая безопасный и упорядоченный доступ к общим ресурсам. Механизмы синхронизации, включая мьютексы, семафоры, события и барьеры, предоставляют инструменты для решения задач синхронизации в различных сценариях. Эти подходы позволяют поддерживать консистентность данных, избегать конфликтов и эффективно управлять многопоточными и многопроцессорными системами, что делает их важной частью разработки высоконагруженных приложений и систем реального времени.

# ЗАКЛЮЧЕНИЕ

В многопоточных и многопроцессорных системах одной из основных задач является обеспечение взаимного исключения и синхронизации процессов и потоков. Эти механизмы необходимы для корректного взаимодействия между потоками, которые одновременно обращаются к общим ресурсам, таким как память, файлы или внешние устройства. Без надлежащего управления доступом могут возникнуть ситуации гонок, некорректные изменения данных или их потеря, что приведёт к сбоям в работе системы.

Критический ресурс — это объект с ограниченным доступом, а критическая секция — участок кода, работающий с этим ресурсом. Ограничение одновременного доступа к таким участкам позволяет избежать несогласованности данных. Однако для реализации этого механизма необходимо обеспечить атомарность операций, особенно при проверке и изменении общих объектов. Отсутствие атомарности может привести к нарушению согласованности, особенно при работе с глобальными переменными, общими структурами данных или другими разделяемыми ресурсами.

Применение механизмов синхронизации, таких как мьютексы, семафоры, события и барьеры, а также функций ожидания, предоставляет разработчикам эффективные инструменты для управления потоками. Каждый из этих механизмов обладает своими особенностями, что позволяет выбирать их в зависимости от конкретных требований задачи. Например, мьютексы обеспечивают строгое взаимное исключение, семафоры позволяют контролировать доступ к ресурсам с ограниченной пропускной способностью, а события используются для уведомления потоков о наступлении определённых условий.

Кроме того, использование спин-блокировок в высокопроизводительных приложениях помогает минимизировать затраты на переключение контекста потоков, что особенно полезно в системах с низкими задержками. Барьеры обеспечивают синхронизацию больших групп потоков, что особенно актуально в задачах, связанных с параллельной обработкой данных.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Critical Section Objects [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/sync/critical-section-objects

[2] Wait Functions – Win32 Apps [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/sync/wait-functions

[3] Synchronization barriers [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/sync/synchronization-barriers

# ПРИЛОЖЕНИЕ А

# (обязательное)

# Исходный код программы

#include <iostream>

#include <vector>

#include <thread>

#include <mutex>

#include <shared\_mutex>

#include <condition\_variable>

#include <chrono>

#include <random>

const size\_t TOTAL\_MEMORY\_SIZE = 1024 \* 1024;

const size\_t BLOCK\_SIZE = 1024;

const size\_t NUM\_BLOCKS = TOTAL\_MEMORY\_SIZE / BLOCK\_SIZE;

const size\_t NUM\_READERS = 5;

const size\_t NUM\_WRITERS = 5;

const size\_t NUM\_OPERATIONS = 20;

struct MemoryBlock {

std::vector<char> data;

std::shared\_mutex mtx;

MemoryBlock() : data(BLOCK\_SIZE, 0) {}

};

std::vector<MemoryBlock> sharedMemory(NUM\_BLOCKS);

void readerFunction(int readerId) {

std::default\_random\_engine generator(readerId);

std::uniform\_int\_distribution<int> blockDistribution(0, NUM\_BLOCKS - 1);

for (size\_t i = 0; i < NUM\_OPERATIONS; ++i) {

int blockIndex = blockDistribution(generator);

{

std::shared\_lock<std::shared\_mutex> lock(sharedMemory[blockIndex].mtx);

char value = sharedMemory[blockIndex].data[0];

std::this\_thread::sleep\_for(std::chrono::milliseconds(10));

std::cout << "Reader " << readerId << " read from block " << blockIndex << "\n";

}

std::this\_thread::sleep\_for(std::chrono::milliseconds(15));

}

}

void writerFunction(int writerId) {

std::default\_random\_engine generator(writerId + 100);

std::uniform\_int\_distribution<int> blockDistribution(0, NUM\_BLOCKS - 1);

for (size\_t i = 0; i < NUM\_OPERATIONS; ++i) {

int blockIndex = blockDistribution(generator);

{

std::unique\_lock<std::shared\_mutex> lock(sharedMemory[blockIndex].mtx);

sharedMemory[blockIndex].data[0] = static\_cast<char>(writerId);

std::this\_thread::sleep\_for(std::chrono::milliseconds(10));

std::cout << "Writer " << writerId << " wrote to block " << blockIndex << "\n";

}

std::this\_thread::sleep\_for(std::chrono::milliseconds(15));

}

}

int main() {

std::vector<std::thread> readers;

for (int i = 0; i < NUM\_READERS; ++i) {

readers.emplace\_back(readerFunction, i);

}

std::vector<std::thread> writers;

for (int i = 0; i < NUM\_WRITERS; ++i) {

writers.emplace\_back(writerFunction, i);

}

for (auto& reader : readers) {

reader.join();

}

for (auto& writer : writers) {

writer.join();

}

std::cout << "All readers and writers have finished their operations.\n";

return 0;

}