Лабораторная работа по теме: Атомарные операции, замена значения переменной и условная замена значения переменной

Цель работы:

* Изучить основные принципы атомарных операций.
* Освоить работу с атомарными операциями для замены значения переменной.
* Освоить применение условной замены значения переменной.
* Научиться работать с функциями атомарных операций в C++.

Ход работы:

Этап 1: Замена значения переменной

Демонстрация использования атомарной функции InterlockedExchange, которая заменяет одно значение переменной на другое.

Код:

#include <iostream>

#include <windows.h>

#include <io.h>

#include <fcntl.h>

int main() {

// Настройка консоли для вывода Unicode

\_setmode(\_fileno(stdout), \_O\_U16TEXT);

LONG sharedVariable = 10; // Переменная, с которой работают несколько потоков

LONG newValue = 20;

// Атомарная замена значения переменной

LONG previousValue = InterlockedExchange(&sharedVariable, newValue);

std::wcout << L"Предыдущее значение: " << previousValue << std::endl;

std::wcout << L"Новое значение: " << sharedVariable << std::endl;

return 0;

}

Результат:



Этап 2: Условная замена значения переменной

Демонстрация использования атомарной функции InterlockedCompareExchange, которая заменяет значение переменной только в том случае, если текущее значение соответствует ожидаемому.

Код:

#include <iostream>

#include <windows.h>

#include <io.h>

#include <fcntl.h>

int main() {

// Настройка консоли для вывода Unicode

\_setmode(\_fileno(stdout), \_O\_U16TEXT);

LONG sharedVariable = 10; // Переменная для работы

LONG newValue = 20;

LONG expectedValue = 10;

// Условная замена значения переменной

LONG previousValue = InterlockedCompareExchange(&sharedVariable, newValue, expectedValue);

if (previousValue == expectedValue) {

std::wcout << L"Значение успешно изменено на: " << sharedVariable << std::endl;

}

else {

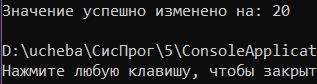
std::wcout << L"Значение не изменено. Текущее значение: " << sharedVariable << std::endl;

}

return 0;

}

Результат:



Если изменить текущее значение с 10 на 8:



Самостоятельные задания

Задание 1:

Модифицируйте программу так, чтобы несколько потоков одновременно пытались выполнить замену значения переменной с использованием InterlockedExchange.

Код:

#include <iostream>

#include <windows.h>

#include <io.h>

#include <fcntl.h>

LONG sharedVariable = 10; // Общая переменная для потоков

HANDLE consoleMutex; // Мьютекс для синхронизации доступа к консоли

DWORD WINAPI ThreadFunction(LPVOID lpParam) {

LONG newValue = (LONG)(lpParam); // Новое значение для потока

// Атомарная замена значения

LONG previousValue = InterlockedExchange(&sharedVariable, newValue);

// Ожидание доступа к консоли

WaitForSingleObject(consoleMutex, INFINITE);

// Синхронизированный вывод информации

std::wcout << L"Поток " << GetCurrentThreadId()

<< L" заменил значение с " << previousValue

<< L" на " << sharedVariable << std::endl;

// Освобождение мьютекса

ReleaseMutex(consoleMutex);

return 0;

}

int main() {

// Настройка консоли для вывода Unicode

\_setmode(\_fileno(stdout), \_O\_U16TEXT);

// Инициализация мьютекса

consoleMutex = CreateMutex(NULL, FALSE, NULL);

HANDLE threads[3];

// Создание трех потоков

for (int i = 0; i < 3; ++i) {

threads[i] = CreateThread(NULL, 0, ThreadFunction, (LPVOID)(i + 1), 0, NULL);

}

// Ожидание завершения всех потоков

WaitForMultipleObjects(3, threads, TRUE, INFINITE);

// Закрытие мьютекса

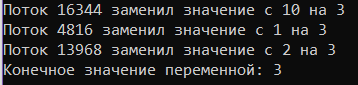
CloseHandle(consoleMutex);

std::wcout << L"Конечное значение переменной: " << sharedVariable << std::endl;

return 0;

}

Результат:



Задание 2:

Используя InterlockedCompareExchange, реализуйте код, в котором несколько потоков пытаются изменить переменную, но только один поток сможет это сделать при выполнении условия.

Код:

#include <iostream>

#include <windows.h>

#include <io.h>

#include <fcntl.h>

LONG sharedVariable = 10; // Общая переменная для потоков

DWORD WINAPI ThreadFunction(LPVOID lpParam) {

LONG newValue = (LONG)(lpParam); // Новое значение для потока

LONG expectedValue = 10; // Ожидаемое значение

// Условная замена значения переменной

LONG previousValue = InterlockedCompareExchange(&sharedVariable, newValue, expectedValue);

if (previousValue == expectedValue) {

std::wcout << L"Поток " << GetCurrentThreadId() << L" успешно заменил значение на " << sharedVariable << std::endl;

}

else {

std::wcout << L"Поток " << GetCurrentThreadId() << L" не смог изменить значение. Текущее значение: " << sharedVariable << std::endl;

}

return 0;

}

int main() {

// Настройка консоли для вывода Unicode

\_setmode(\_fileno(stdout), \_O\_U16TEXT);

HANDLE threads[3];

// Создание трех потоков

for (int i = 0; i < 3; ++i) {

threads[i] = CreateThread(NULL, 0, ThreadFunction, (LPVOID)(i + 1), 0, NULL);

}

// Ожидание завершения всех потоков

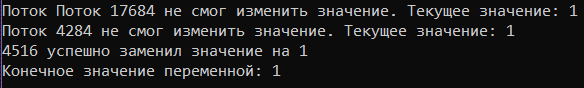
WaitForMultipleObjects(3, threads, TRUE, INFINITE);

std::wcout << L"Конечное значение переменной: " << sharedVariable << std::endl;

return 0;

}

Результат:



Контрольные вопросы:

1. Что такое атомарные операции?

Атомарные операции — это такие операции, которые выполняются полностью, без возможности прерывания.

1. В каких случаях применяются атомарные операции?

Атомарные операции применяются в многопоточных средах для предотвращения конфликтов при одновременном доступе к данным несколькими потоками, особенно для изменения общих переменных.

1. Как работает InterlockedExchange?

Эта функция атомарно заменяет одно значение переменной на другое. Это полезно, когда требуется гарантировать, что несколько потоков не смогут одновременно изменить переменную и привести к непредсказуемым результатам.

1. Чем отличается InterlockedExchange от InterlockedCompareExchange?

InterlockedExchange заменяет значение переменной независимо от её текущего состояния, тогда как InterlockedCompareExchange заменяет значение только при условии, что текущее значение соответствует ожидаемому.

1. Какую роль играют атомарные операции в многопоточных программах?

Атомарные операции играют ключевую роль в обеспечении безопасности данных, предотвращая состояния гонки и гарантируя, что изменения переменных происходят корректно и последовательно.

1. Какие ошибки могут возникнуть при отсутствии атомарности?

При отсутствии атомарности могут возникнуть состояния гонки, потеря данных, некорректные результаты вычислений и другие проблемы, связанные с одновременным доступом к данным.

1. Как работает условная замена значения переменной?

Условная замена значения переменной (InterlockedCompareExchange) заменяет значение переменной на новое только в том случае, если текущее значение соответствует ожидаемому. Если оно не совпадает, замена не происходит.

1. Можно ли использовать атомарные операции для работы с типами данных больше, чем LONG?

В большинстве языков программирования атомарные операции, как правило, поддерживают только стандартные типы данных, такие как LONG или указатели. Для более сложных типов данных могут потребоваться дополнительные механизмы синхронизации.

1. Чем опасны неатомарные операции в многопоточной среде?

Неатомарные операции могут привести к состояниям гонки, потере данных, некорректным результатам и другим ошибкам, которые сложно отладить и воспроизвести.

1. Какие функции для атомарных операций предоставляет C++?

В C++ для атомарных операций используются функции из библиотеки <atomic>, такие как std::atomic, std::atomic\_exchange, std::atomic\_compare\_exchange, и другие, которые обеспечивают атомарность и безопасность при работе с многопоточностью.