**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++.

**Теоретический материал:**

**Атомарные операции** — это такие операции, которые выполняются полностью, без возможности прерывания. Это особенно важно в многопоточных программах, где несколько потоков могут одновременно изменять одни и те же данные. Если операция атомарна, это значит, что она выполняется как единое действие, и другой поток не может вмешаться в процесс изменения данных.

**Почему важны атомарные операции?**  
В многопоточной среде обычные операции могут быть небезопасными для данных, так как несколько потоков могут одновременно читать и записывать данные. Это приводит к таким проблемам, как **состояния гонки** (race conditions), когда результат программы зависит от того, какой поток первым завершит выполнение. Атомарные операции помогают избежать таких ситуаций, гарантируя, что изменение переменной произойдет полностью до того, как другой поток сможет вмешаться.

**Основные атомарные операции**

1. **Замена значения переменной** (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.
   * Операция работает как единая, непрерываемая, и таким образом гарантирует целостность данных.
2. **Условная замена значения переменной** (InterlockedCompareExchange):
   * Эта функция атомарно заменяет значение переменной только в том случае, если текущее значение соответствует ожидаемому. Если текущее значение не соответствует ожидаемому, замена не происходит.
   * Это полезно для предотвращения гонок состояний, когда несколько потоков могут одновременно попытаться изменить одну и ту же переменную.

**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;

}

**Объяснение:**

* InterlockedExchange атомарно заменяет значение переменной sharedVariable с 10 на 20.
* Функция возвращает предыдущее значение переменной до замены.

**Ожидаемый результат:**

Предыдущее значение: 10

Новое значение: 20

**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;

}

**Объяснение:**

* InterlockedCompareExchange заменяет значение переменной на 20 только в том случае, если текущее значение равно 10 (значению expectedValue).
* Если текущее значение не совпадает с ожидаемым, замена не произойдет.

**Ожидаемый результат:**   
Если текущее значение совпадает с ожидаемым: Значение успешно изменено на: 20  
Если оно не совпадает, то: Значение не изменено. Текущее значение: 10

Дополнительно выполните код с другим значением и добавьте результат в отчет

**Самостоятельные задания**

**Задание 1:**

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

**Подсказка:**  
Добавьте следующий код в функцию main, чтобы создать несколько потоков, которые будут работать с общей переменной. Каждый поток будет пытаться заменить значение переменной на своё значение.

**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;

}

**Ожидаемый результат:** Каждый поток будет заменять значение переменной, и в результате вывод будет показывать, как меняется значение переменной в каждом потоке.

**Задание 2:**

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

**Подсказка:**  
Добавьте код, который создаёт несколько потоков. Каждый поток пытается выполнить условную замену значения переменной с помощью InterlockedCompareExchange.

**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;

}

**Ожидаемый результат:** Один из потоков изменит значение переменной на новое, если текущее значение будет равно ожидаемому.

**Контрольные вопросы:**

1. Что такое атомарные операции?
2. В каких случаях применяются атомарные операции?
3. Как работает InterlockedExchange?
4. Чем отличается InterlockedExchange от InterlockedCompareExchange?
5. Какую роль играют атомарные операции в многопоточных программах?
6. Какие ошибки могут возникнуть при отсутствии атомарности?
7. Как работает условная замена значения переменной?
8. Можно ли использовать атомарные операции для работы с типами данных больше, чем LONG?
9. Чем опасны неатомарные операции в многопоточной среде?
10. Какие функции для атомарных операций предоставляет C++?