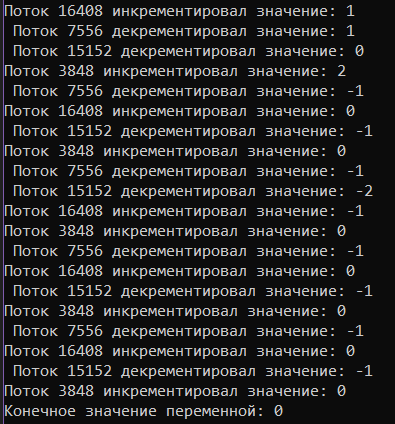
Лабораторная работа: Инкремент и декремент переменной, изменение значений переменной

Цели и задания

Научиться использовать атомарные операции для инкремента и декремента переменной.

Изучить методы изменения значений переменной в многопоточном окружении.

Реализовать многопоточную программу, которая демонстрирует инкремент и декремент переменной.



Самостоятельные задания

Задание 1: Измените программу так, чтобы она инкрементировала переменную на 2 вместо 1. Используйте InterlockedAdd для этой операции.

Подсказка: Замените InterlockedIncrement на InterlockedAdd с добавлением 2.

#include <iostream>

#include <windows.h>

#include <io.h>

#include <fcntl.h>

LONG sharedVariable = 0; // Общая переменная для потоков

DWORD WINAPI IncrementThreadFunction(LPVOID lpParam) {

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

LONG incrementedValue = InterlockedAdd(&sharedVariable, 2); // Инкремент на 2

std::wcout << L"Поток " << GetCurrentThreadId() << L" инкрементировал значение: " << incrementedValue << std::endl;

Sleep(100); // Задержка для демонстрации

}

return 0;

}

DWORD WINAPI DecrementThreadFunction(LPVOID lpParam) {

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

LONG decrementedValue = InterlockedDecrement(&sharedVariable);

std::wcout << L"Поток " << GetCurrentThreadId() << L" декрементировал значение: " << decrementedValue << std::endl;

Sleep(100); // Задержка для демонстрации

}

return 0;

}

int main() {

// Настройка консоли для вывода

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

HANDLE incrementThreads[2];

HANDLE decrementThreads[2];

// Создание потоков для инкремента

for (int i = 0; i < 2; ++i) {

incrementThreads[i] = CreateThread(NULL, 0, IncrementThreadFunction, NULL, 0, NULL);

}

// Создание потоков для декремента

for (int i = 0; i < 2; ++i) {

decrementThreads[i] = CreateThread(NULL, 0, DecrementThreadFunction, NULL, 0, NULL);

}

// Ожидание завершения всех потоков

WaitForMultipleObjects(2, incrementThreads, TRUE, INFINITE);

WaitForMultipleObjects(2, decrementThreads, TRUE, INFINITE);

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

// Закрываем дескрипторы потоков

for (int i = 0; i < 2; ++i) {

CloseHandle(incrementThreads[i]);

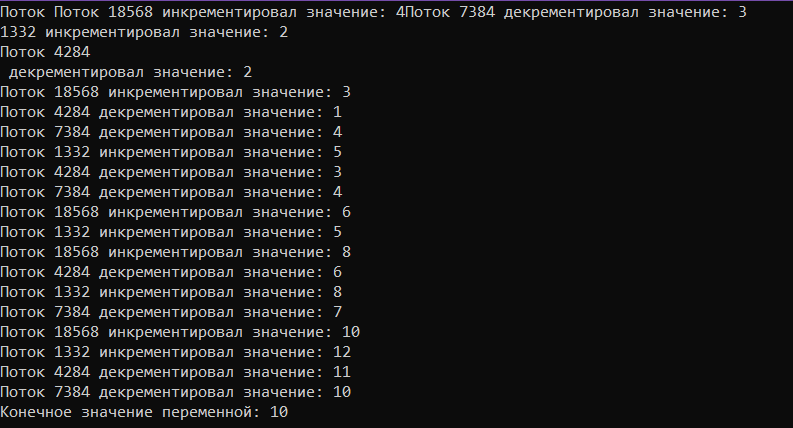
CloseHandle(decrementThreads[i]);

}

return 0;

}

Результат:



Задание 2: Добавьте еще один поток, который будет выполнять инкрементирование переменной и выводить результат.

Подсказка: Создайте новый поток с той же функцией инкрементации.

Код:

#include <iostream>

#include <windows.h>

#include <io.h>

#include <fcntl.h>

LONG sharedVariable = 0; // Общая переменная для потоков

DWORD WINAPI IncrementThreadFunction(LPVOID lpParam) {

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

LONG incrementedValue = InterlockedAdd(&sharedVariable, 2); // Инкремент на 2

std::wcout << L"Поток " << GetCurrentThreadId() << L" инкрементировал значение: " << incrementedValue << std::endl;

Sleep(100); // Задержка для демонстрации

}

return 0;

}

DWORD WINAPI DecrementThreadFunction(LPVOID lpParam) {

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

LONG decrementedValue = InterlockedDecrement(&sharedVariable);

std::wcout << L"Поток " << GetCurrentThreadId() << L" декрементировал значение: " << decrementedValue << std::endl;

Sleep(100); // Задержка для демонстрации

}

return 0;

}

int main() {

// Настройка консоли для вывода Unicode

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

HANDLE incrementThreads[3]; // Увеличиваем количество потоков для инкремента

HANDLE decrementThreads[2];

// Создание потоков для инкремента

for (int i = 0; i < 3; ++i) { // Изменяем на 3 потока

incrementThreads[i] = CreateThread(NULL, 0, IncrementThreadFunction, NULL, 0, NULL);

}

// Создание потоков для декремента

for (int i = 0; i < 2; ++i) {

decrementThreads[i] = CreateThread(NULL, 0, DecrementThreadFunction, NULL, 0, NULL);

}

// Ожидание завершения всех потоков

WaitForMultipleObjects(3, incrementThreads, TRUE, INFINITE); // Изменяем на 3

WaitForMultipleObjects(2, decrementThreads, TRUE, INFINITE);

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

// Закрываем дескрипторы потоков

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

CloseHandle(incrementThreads[i]);

}

for (int i = 0; i < 2; ++i) {

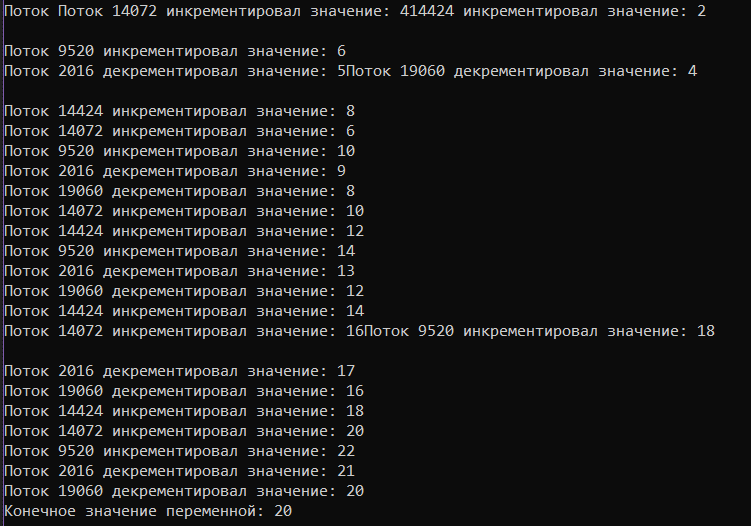
CloseHandle(decrementThreads[i]);

}

return 0;

}

Результат:



Контрольные вопросы

1. Что такое атомарные операции

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

1. Какую роль играют InterlockedIncrement и InterlockedDecrement в многопоточном программировании

InterlockedIncrement и InterlockedDecrement — это атомарные функции, которые безопасно увеличивают или уменьшают значение переменной на единицу. Они предотвращают состояние гонки, позволяя нескольким потокам одновременно изменять общую переменную без риска повреждения данных.

1. Почему важно использовать атомарные операции при работе с общими переменными

Использование атомарных операций позволяет избежать состояния гонки, когда два или более потоков пытаются одновременно изменить одну и ту же переменную. Это гарантирует, что операции над переменной будут выполнены корректно и предсказуемо, обеспечивая целостность данных.

1. Какое значение будет у sharedVariable после выполнения двух потоков инкрементации и двух потоков декрементации

Если два потока инкрементируют переменную на 2 каждый (всего +4) и два потока декрементируют переменную на 1 каждый (всего -2), то итоговое значение sharedVariable будет равно:

sharedVariable = 0 + 4 - 2 = 2

Таким образом, конечное значение будет 2.

1. Как можно предотвратить состояние гонки в многопоточной программе

Состояние гонки можно предотвратить с помощью:

* Атомарных операций (например, InterlockedIncrement, InterlockedDecrement).
* Мьютексов и семафоров для синхронизации доступа к общим ресурсам.
* Использования критических секций для ограничения доступа к общим данным.

1. Какие еще атомарные операции вы знаете

Кроме InterlockedIncrement и InterlockedDecrement, существуют и другие атомарные операции, такие как:

* InterlockedAdd: добавляет значение к переменной.
* InterlockedExchange: заменяет значение переменной и возвращает предыдущее значение.
* InterlockedCompareExchange: сравнивает текущее значение переменной с ожидаемым и, если они равны, заменяет его на новое значение.

1. Что произойдет, если два потока попытаются одновременно изменить одну и ту же переменную без атомарных операций

Если два потока попытаются одновременно изменить одну и ту же переменную без атомарных операций, это может привести к состоянию гонки. В результате переменная может оказаться в неконсистентном состоянии, и ее значение может быть неправильным или неожиданным, так как изменения одного потока могут перезаписать изменения другого.

1. Как вы можете вывести текущее значение переменной в конце работы программы

Для вывода текущего значения переменной в конце работы программы можно использовать стандартный вывод, как в примере:

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

1. В чем разница между инкрементом и декрементом переменной

Инкремент — это операция, которая увеличивает значение переменной на единицу (или на заданное значение), в то время как декремент — это операция, которая уменьшает значение переменной на единицу (или на заданное значение). Например, инкрементирование переменной x на 1 будет x++, а декрементирование — x--.

1. Как вы можете протестировать свою программу на наличие ошибок?

Для тестирования программы на наличие ошибок в многопоточной среде можно использовать:

* Написание тестов, которые запускают программу с различными конфигурациями потоков и проверяют корректность результатов.
* Использование инструментов для обнаружения гонок данных, таких как ThreadSanitizer.
* Проведение стресс-тестирования, где программа будет работать под высокой нагрузкой, чтобы выявить возможные проблемы в синхронизации и управлении потоками.