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 №6

на тему

Средства синхронизации и взаимного исключения (*Windows*). Изучение и использование средств синхронизации и взаимного исключения

Выполнил:

студент гр. 153503

Звягинцева В.А.

Проверил:

Гриценко Н. Ю.

Минск 2023

СОДЕРЖАНИЕ

1 Постановка задачи3

2 Краткие теоретические сведения4

2.1 Объект мьютекса4

2.2 Объект семафора5

3 Результаты выполнения лабораторной работы7

Вывод8

Список использованных источников9

Приложение А10

**1 ПОСТАНОВКА ЗАДАЧИ**

Целью выполнения лабораторной работы является разработка приложения для синхронизации доступа разных потоков к одной переменной при помощи мьютекса.

**2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

**2.1 Объект мьютекса**

Объект мьютекса — это объект синхронизации, состояние которого устанавливается в значение *Signaled*, если он не принадлежит ни одному потоку, и без знака, когда он принадлежит. Только один поток за раз может владеть объектом мьютекса, имя которого происходит из-за того, что он полезен для координации взаимоисключающего доступа к общему ресурсу. Например, чтобы предотвратить запись двух потоков в общую память одновременно, каждый поток ожидает владения объектом мьютекса перед выполнением кода, который обращается к памяти. После записи в общую память поток освобождает объект мьютекса.

Поток использует функцию *CreateMutex* или *CreateMutexEx* для создания объекта мьютекса. Создающий поток может запросить немедленное владение объектом мьютекса, а также указать имя для объекта мьютекса. Он также может создать неименованный мьютекс.

Потоки в других процессах могут открывать дескриптор существующего именованного объекта мьютекса, указывая его имя в вызове функции *OpenMutex*. Чтобы передать дескриптор неименованного мьютекса другому процессу, нужно использовать функцию *DuplicateHandle* или наследование дескриптора типа "родители-потомки".

Любой поток с дескриптором объекта мьютекса может использовать одну из функций ожидания для запроса владения объектом мьютекса. Если объект мьютекса принадлежит другому потоку, функция ожидания блокирует запрашивающий поток до тех пор, пока поток-владение не освободит объект мьютекса с помощью функции *ReleaseMutex*. Возвращаемое значение функции *wait* указывает, была ли функция возвращена по какой-либо причине, кроме состояния мьютекса, для которой задано значение *signaled*.

Если для мьютекса ожидается несколько потоков, выбирается ожидающий поток. Внешние события, такие как *APC* в режиме ядра, могут изменить порядок ожидания.

После того как поток получает право владения мьютексом, он может указать тот же мьютекс в неоднократных вызовах функций ожидания, не блокируя его выполнение. Это предотвращает взаимоблокировку потока при ожидании мьютекса, которым он уже владеет. Чтобы освободить своё владение в таких обстоятельствах, поток должен вызывать *ReleaseMutex* один раз для каждого случая, когда мьютекс удовлетворяет условиям функции ожидания.

Если поток завершается, не освобождая права владения объектом мьютекса, объект мьютекса считается заброшенным. Ожидающий поток может получить владение заброшенным объектом мьютекса, но функция ожидания вернёт *WAIT\_ABANDONED*, чтобы указать, что объект мьютекса заброшен. Заброшенный объект мьютекса указывает, что произошла ошибка и что любой общий ресурс, защищённый объектом мьютекса, находится в неопределённом состоянии. Если поток продолжает работу так, как будто объект мьютекса не был оставлен, он больше не считается брошенным после того, как поток освобождается от владения. Это восстанавливает нормальное поведение, если дескриптор объекта мьютекса впоследствии указывается в функции ожидания. [1]

**2.2 Объект семафора**

Объект семафора — это объект синхронизации, который поддерживает число от нуля до указанного максимального значения. Число уменьшается каждый раз, когда поток завершает ожидание объекта семафора, и увеличивается каждый раз, когда поток освобождает семафор. Когда число достигает нуля, больше потоки не могут ждать, пока состояние объекта семафора станет сигналом. Состояние семафора становится сигнальным, когда это число становится больше нуля, и несигнальным, когда равно нулю.

Объект семафора полезен при управлении общим ресурсом, который может поддерживать ограниченное число пользователей. Он выступает в качестве шлюза, который ограничивает количество потоков, совместно использующих ресурс, указанным максимальным числом. Например, приложение может установить ограничение на количество создаваемых окон. Он использует семафор с максимальным числом, равным предельному значению окна, уменьшая счётчик при создании окна и при каждом закрытии окна. Приложение указывает объект семафора в вызове одной из функций ожидания перед созданием каждого окна. Если значение счётчика равно нулю, указывающее, что достигнуто предельное количество окон, функция ожидания блокирует выполнение кода создания окна.

Поток использует функцию *CreateSemaphore* или *CreateSemaphoreEx* для создания объекта семафора. В потоке создания указывается начальное число и максимальное значение счётчика для объекта. Начальное число не должно быть ни меньше нуля, ни больше максимального значения. Создающий поток также может указать имя объекта семафора. Потоки в других процессах могут открывать дескриптор существующего объекта семафора, указывая его имя в вызове функции *OpenSemaphore*.

Если на семафоре ожидается несколько потоков, выбирается ожидающий поток. Внешние события, такие как *APC* в режиме ядра, могут изменить порядок ожидания.

Каждый раз, когда одна из функций ожидания возвращает значение, так как состояние семафора было задано как сигнальное, количество семафора уменьшается на один. Функция *ReleaseSemaphore* увеличивает количество семафоров на указанную величину. Число никогда не может быть меньше нуля или больше максимального значения.

Начальное число семафора обычно устанавливается в максимальное значение. Затем количество уменьшается по сравнению с этим уровнем по мере использования защищённого ресурса. Кроме того, можно создать семафор с начальным числом, равным нулю, чтобы заблокировать доступ к защищённому ресурсу во время инициализации приложения. После инициализации можно использовать *ReleaseSemaphore*, чтобы увеличить число до максимального значения.

Поток, владеющий объектом мьютекса, может несколько раз ожидать передачи сигнала об этом же объекте мьютекса без блокировки его выполнения. Однако поток, который несколько раз ожидает одного и того же объекта семафора, уменьшает число семафоров при каждом завершении операции ожидания; поток блокируется, когда счётчик становится равным нулю. Аналогичным образом только поток, владеющий мьютексом, может успешно вызывать функцию *ReleaseMutex*, хотя любой поток может использовать *ReleaseSemaphore* для увеличения числа объектов семафора.

Поток может уменьшать количество семафоров несколько раз, многократно указывая один и тот же объект семафора в вызовах любой из функций ожидания. Однако вызов одной из функций ожидания с несколькими объектами с массивом, содержащим несколько дескрипторов одного семафора, не приводит к нескольким уменьшениям.

Завершив использование объекта семафора, нужно вызвать функцию *CloseHandle*, чтобы закрыть дескриптор. Объект семафора уничтожается при закрытии последнего дескриптора. Закрытие дескриптора не влияет на число семафоров, поэтому обязательно нужно вызвать *ReleaseSemaphore* перед закрытием дескриптора или перед завершением процесса. В противном случае время ожидания операций ожидания будет истекает или продолжается бесконечно в зависимости от того, задано ли значение времени ожидания. [2]

**3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ**

В ходе выполнения лабораторной работы было разработано приложение для синхронизации доступа разных потоков к одной переменной при помощи мьютекса (рисунок 3.1).

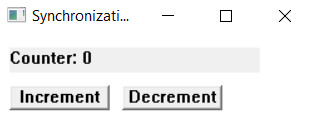


Рисунок 3.1. – Результат работы программы

В главном окне есть лейбл для отображения значения счётчика и кнопки инкремента и декремента. При нажатии на кнопку инкремента значение счётчика увеличивается (рисунок 3.2).

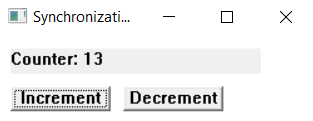


Рисунок 3.2. – Работа программы при нажатии кнопки инкремента

При нажатии на кнопку декремента значение счётчика уменьшается (рисунок 3.3).

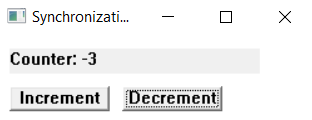


Рисунок 3.3 – Работа программы при нажатии кнопки декремента

**ВЫВОД**

В результате выполнения лабораторной работы были изучены средства синхронизации и взаимного исключения (*Windows*). Было создано приложение для синхронизации доступа разных потоков к одной переменной при помощи мьютекса.

**СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ**

1. Использование объектов мьютекса [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/sync/using-mutex-objects.
2. Использование объектов семафора [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/sync/using-semaphore-objects.

**ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Листинг кода**

**main.cpp**

#include <windows.h>

#include <tchar.h>

#include <thread>

HANDLE hMutex; // Дескриптор мьютекса

int counter = 0; // Общая переменная

HWND hCounterLabel, hIncrementButton, hDecrementButton;

#define WM\_UPDATE\_COUNTER (WM\_USER + 1) // Сообщение для обновления счетчика

void incrementCounter() {

while (true) {

WaitForSingleObject(hMutex, INFINITE); // Захват мьютекса

counter++;

ReleaseMutex(hMutex); // Освобождение мьютекса

PostMessage(hCounterLabel, WM\_UPDATE\_COUNTER, 0, 0);

std::this\_thread::sleep\_for(std::chrono::milliseconds(500)); // Задержка для имитации работы

}

}

void decrementCounter() {

while (true) {

WaitForSingleObject(hMutex, INFINITE);

counter--;

ReleaseMutex(hMutex);

PostMessage(hCounterLabel, WM\_UPDATE\_COUNTER, 0, 0);

std::this\_thread::sleep\_for(std::chrono::milliseconds(500));

}

}

void updateCounterLabel() {

TCHAR buffer[256];

\_stprintf\_s(buffer, \_T("Counter: %d"), counter);

SetWindowText(hCounterLabel, buffer);

}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam) {

switch (message) {

case WM\_CREATE:

hCounterLabel = CreateWindow(\_T("STATIC"), \_T(""), WS\_VISIBLE | WS\_CHILD, 10, 10, 200, 20, hWnd, nullptr, nullptr, nullptr);

hIncrementButton = CreateWindow(\_T("BUTTON"), \_T("Increment"), WS\_VISIBLE | WS\_CHILD, 10, 40, 80, 20, hWnd, nullptr, nullptr, nullptr);

hDecrementButton = CreateWindow(\_T("BUTTON"), \_T("Decrement"), WS\_VISIBLE | WS\_CHILD, 100, 40, 80, 20, hWnd, nullptr, nullptr, nullptr);

updateCounterLabel();

// Создание мьютекса

hMutex = CreateMutex(nullptr, FALSE, nullptr);

// Запуск потоков в отдельном потоке

std::thread(incrementCounter).detach();

std::thread(decrementCounter).detach();

break;

case WM\_CLOSE:

DestroyWindow(hWnd);

break;

case WM\_DESTROY:

CloseHandle(hMutex); // Закрытие мьютекса при завершении приложения

PostQuitMessage(0);

break;

case WM\_UPDATE\_COUNTER:

updateCounterLabel();

break;

case WM\_COMMAND:

if (lParam == (LPARAM)hIncrementButton) {

WaitForSingleObject(hMutex, INFINITE);

counter++;

ReleaseMutex(hMutex);

updateCounterLabel();

}

else if (lParam == (LPARAM)hDecrementButton) {

WaitForSingleObject(hMutex, INFINITE);

counter--;

ReleaseMutex(hMutex);

updateCounterLabel();

}

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

return 0;

}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow) {

WNDCLASSEX wcex = { sizeof(WNDCLASSEX) };

wcex.lpfnWndProc = WndProc;

wcex.hInstance = hInstance;

wcex.hCursor = LoadCursor(nullptr, IDC\_ARROW);

wcex.lpszClassName = \_T("SyncApp");

RegisterClassEx(&wcex);

HWND hWnd = CreateWindow(\_T("SyncApp"), \_T("Synchronization Example"), WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, CW\_USEDEFAULT, 300, 150, nullptr, nullptr, hInstance, nullptr);

if (!hWnd) {

return false;

}

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

MSG msg;

while (GetMessage(&msg, nullptr, 0, 0)) {

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return static\_cast<int>(msg.wParam);

}