Лабораторная работа: Семафоры в Windows

Цель работы:

Изучить основы синхронизации потоков с использованием семафоров в операционной системе Windows, научиться контролировать доступ к общим ресурсам и управлять параллельным выполнением задач.

Код 1:

#include <iostream>

#include <windows.h>

#include <io.h>

#include <fcntl.h>

HANDLE semaphore; // Глобальная переменная для семафора

DWORD WINAPI ThreadFunction(LPVOID lpParam) {

std::wcout << L"Поток " << GetCurrentThreadId() << L" ожидает доступ к ресурсу...\n";

// Ожидание освобождения семафора

WaitForSingleObject(semaphore, INFINITE);

std::wcout << L"Поток " << GetCurrentThreadId() << L" получил доступ к ресурсу.\n";

Sleep(2000); // Имитация работы с ресурсом

// Освобождаем семафор

ReleaseSemaphore(semaphore, 1, NULL);

std::wcout << L"Поток " << GetCurrentThreadId() << L" завершил работу с ресурсом.\n";

return 0;

}

int main() {

// Устанавливаем кодировку консоли для вывода Unicode символов

SetConsoleOutputCP(CP\_UTF8);

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

// Создаем семафор с начальным значением 2

semaphore = CreateSemaphore(NULL, 2, 2, NULL);

if (semaphore == NULL) {

std::wcerr << L"Ошибка создания семафора: " << GetLastError() << std::endl;

return 1;

}

HANDLE threads[3];

// Создаём три потока

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

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

}

// Ожидаем завершения всех потоков

WaitForMultipleObjects(3, threads, TRUE, INFINITE);

// Закрываем дескрипторы потоков

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

CloseHandle(threads[i]);

}

// Закрываем семафор

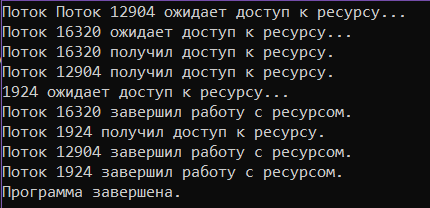
CloseHandle(semaphore);

std::wcout << L"Программа завершена.\n";

return 0;

}

Результат:



Код 2:

#include <iostream>

#include <windows.h>

#include <io.h>

#include <fcntl.h>

#include <stdio.h>

HANDLE semaphore; // Глобальная переменная для семафора

FILE\* file; // Общий файл для записи

DWORD WINAPI ThreadFunction(LPVOID lpParam) {

std::wcout << L"Поток " << GetCurrentThreadId() << L" ожидает доступ к файлу...\n";

// Ожидание доступа к файлу через семафор

WaitForSingleObject(semaphore, INFINITE);

std::wcout << L"Поток " << GetCurrentThreadId() << L" получил доступ к файлу.\n";

// Записываем данные в файл

fwprintf(file, L"Поток %d записывает данные в файл.\n", GetCurrentThreadId());

fflush(file);

Sleep(2000); // Имитация работы с файлом

// Освобождаем семафор

ReleaseSemaphore(semaphore, 1, NULL);

std::wcout << L"Поток " << GetCurrentThreadId() << L" завершил работу с файлом.\n";

return 0;

}

int main() {

// Устанавливаем кодировку консоли для вывода Unicode символов

SetConsoleOutputCP(CP\_UTF8);

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

// Открываем файл для записи безопасным методом

if (\_wfopen\_s(&file, L"output.txt", L"w, ccs=UTF-8") != 0) {

std::wcerr << L"Не удалось открыть файл.\n";

return 1;

}

// Создаем семафор с начальным значением 1 (один поток может работать с файлом)

semaphore = CreateSemaphore(NULL, 1, 1, NULL);

if (semaphore == NULL) {

std::wcerr << L"Ошибка создания семафора: " << GetLastError() << std::endl;

return 1;

}

HANDLE threads[3];

// Создаём три потока

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

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

}

// Ожидаем завершения всех потоков

WaitForMultipleObjects(3, threads, TRUE, INFINITE);

// Закрываем дескрипторы потоков и файл

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

CloseHandle(threads[i]);

}

CloseHandle(semaphore);

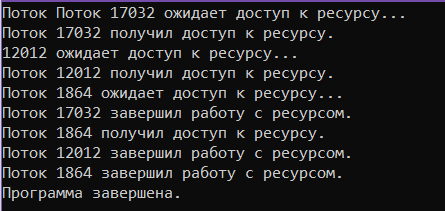
fclose(file);

std::wcout << L"Программа завершена.\n";

return 0;

}

Результат:



Самостоятельные задания:

**Задание 1**: Модифицируйте программу для работы с четырьмя потоками и начальным значением семафора 3.

#include <iostream>

#include <windows.h>

#include <io.h>

#include <fcntl.h>

#include <stdio.h>

HANDLE semaphore; // Глобальная переменная для семафора

FILE\* file; // Общий файл для записи

DWORD WINAPI ThreadFunction(LPVOID lpParam) {

std::wcout << L"Поток " << GetCurrentThreadId() << L" ожидает доступ к файлу...\n";

// Ожидание доступа к файлу через семафор

WaitForSingleObject(semaphore, INFINITE);

std::wcout << L"Поток " << GetCurrentThreadId() << L" получил доступ к файлу.\n";

// Записываем данные в файл

fwprintf(file, L"Поток %d записывает данные в файл.\n", GetCurrentThreadId());

fflush(file);

Sleep(2000); // Имитация работы с файлом

// Освобождаем семафор

ReleaseSemaphore(semaphore, 1, NULL);

std::wcout << L"Поток " << GetCurrentThreadId() << L" завершил работу с файлом.\n";

return 0;

}

int main() {

// Устанавливаем кодировку консоли для вывода Unicode символов

SetConsoleOutputCP(CP\_UTF8);

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

// Открываем файл для записи безопасным методом

if (\_wfopen\_s(&file, L"output.txt", L"w, ccs=UTF-8") != 0) {

std::wcerr << L"Не удалось открыть файл.\n";

return 1;

}

// Создаем семафор с начальным значением 3 (три потока могут работать с файлом)

semaphore = CreateSemaphore(NULL, 3, 3, NULL);

if (semaphore == NULL) {

std::wcerr << L"Ошибка создания семафора: " << GetLastError() << std::endl;

return 1;

}

HANDLE threads[4];

// Создаём четыре потока

for (int i = 0; i < 4; ++i) {

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

}

// Ожидаем завершения всех потоков

WaitForMultipleObjects(4, threads, TRUE, INFINITE);

// Закрываем дескрипторы потоков и файл

for (int i = 0; i < 4; ++i) {

CloseHandle(threads[i]);

}

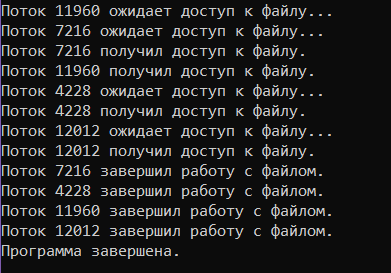
CloseHandle(semaphore);

fclose(file);

std::wcout << L"Программа завершена.\n";

return 0;

}



**Задание 2**: Реализуйте синхронизацию потоков при работе с разными файлами.

* Подсказка: Создайте массив дескрипторов файлов, и каждому потоку назначьте свой файл.

Код:

#include <iostream>

#include <windows.h>

#include <io.h>

#include <fcntl.h>

#include <stdio.h>

HANDLE semaphore; // Глобальная переменная для семафора

FILE\* files[4]; // Массив файлов для записи

DWORD WINAPI ThreadFunction(LPVOID lpParam) {

int threadIndex = \*(int\*)lpParam; // Получаем индекс потока

std::wcout << L"Поток " << GetCurrentThreadId() << L" ожидает доступ к файлу...\n";

// Ожидание доступа к файлу через семафор

WaitForSingleObject(semaphore, INFINITE);

std::wcout << L"Поток " << GetCurrentThreadId() << L" получил доступ к файлу.\n";

// Записываем данные в файл

fwprintf(files[threadIndex], L"Поток %d записывает данные в файл %d.\n", GetCurrentThreadId(), threadIndex);

fflush(files[threadIndex]);

Sleep(2000); // Имитация работы с файлом // Освобождаем семафор

ReleaseSemaphore(semaphore, 1, NULL);

std::wcout << L"Поток " << GetCurrentThreadId() << L" завершил работу с файлом.\n";

return 0;

}

int main() {

// Устанавливаем кодировку консоли для вывода Unicode символов

SetConsoleOutputCP(CP\_UTF8);

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

// Создаем семафор с начальным значением 3

semaphore = CreateSemaphore(NULL, 3, 3, NULL);

if (semaphore == NULL) {

std::wcerr << L"Ошибка создания семафора: " << GetLastError() << std::endl;

return 1;

}

HANDLE threads[4]; // Изменено на 4 потока

int threadIndices[4]; // Массив индексов потоков // Открываем файлы для записи

for (int i = 0; i < 4; ++i) {

wchar\_t filename[20];

swprintf(filename, 20, L"output\_%d.txt", i);

if (\_wfopen\_s(&files[i], filename, L"w, ccs=UTF-8") != 0) {

std::wcerr << L"Не удалось открыть файл " << filename << L".\n";

return 1;

}

threadIndices[i] = i; // Запоминаем индекс потока

threads[i] = CreateThread(NULL, 0, ThreadFunction, &threadIndices[i], 0, NULL);

}

// Ожидаем завершения всех потоков

WaitForMultipleObjects(4, threads, TRUE, INFINITE);

// Закрываем дескрипторы потоков и файлов

for (int i = 0; i < 4; ++i) {

CloseHandle(threads[i]);

fclose(files[i]); // Закрываем каждый файл

}

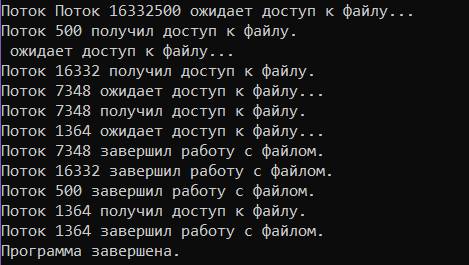
CloseHandle(semaphore);

std::wcout << L"Программа завершена.\n";

return 0;

}

Результат:



Контрольные вопросы

1. Что такое семафор?

Семафор — это механизм синхронизации, который используется для ограничения доступа к общему ресурсу.

1. Для чего используется семафор?

В Windows семафоры могут быть использованы для синхронизации нескольких потоков.

1. Какие параметры принимает функция CreateSemaphore?

Функция CreateSemaphore принимает четыре параметра:

* lpSemaphoreAttributes: указатель на атрибуты безопасности семафора (может быть NULL).
* lInitialCount: начальное значение семафора (количество потоков, которые могут одновременно получить доступ к ресурсу).
* lMaximumCount: максимальное значение семафора (максимальное количество потоков, которые могут одновременно получить доступ к ресурсу).
* lpName: имя семафора (если NULL, семафор доступен только в текущем процессе).

1. Чем отличается WaitForSingleObject от WaitForMultipleObjects?

WaitForSingleObject используется для ожидания завершения одного объекта синхронизации (например, одного семафора или потока), в то время как WaitForMultipleObjects позволяет ожидать завершения нескольких объектов синхронизации одновременно. Это позволяет управлять состоянием нескольких потоков или семафоров в одном вызове.

1. Как освободить семафор?

Для освобождения семафора используется функция ReleaseSemaphore.

1. Что произойдёт, если семафор не будет освобождён?

Если семафор не будет освобождён, потоки, ожидающие его освобождения (с помощью WaitForSingleObject), будут заблокированы и не смогут продолжить выполнение, что может привести к зависанию программы или истощению ресурсов.

1. Сколько потоков могут одновременно получить доступ к ресурсу при начальном значении семафора 3?

При начальном значении семафора 3 одновременно могут получить доступ к ресурсу три потока.

1. Какой код используется для ожидания освобождения ресурса в семафоре?

Для ожидания освобождения ресурса в семафоре используется функция WaitForSingleObject, например:

WaitForMultipleObjects(4, threads, TRUE, INFINITE);

1. В чем преимущества семафоров перед мьютексами?

Преимущества семафоров перед мьютексами заключаются в том, что семафоры могут разрешать доступ нескольким потокам одновременно (в отличие от мьютексов, которые позволяют только одному потоку), что делает их более подходящими для сценариев, где требуется ограниченный параллелизм. Также семафоры могут использоваться для синхронизации между процессами.

1. Как закрыть дескриптор семафора после завершения программы?

Дескриптор семафора закрывается с помощью функции CloseHandle, например:

CloseHandle(semaphore);