Лабораторная работа по теме: Тупики и безопасное завершение тупиков в Windows

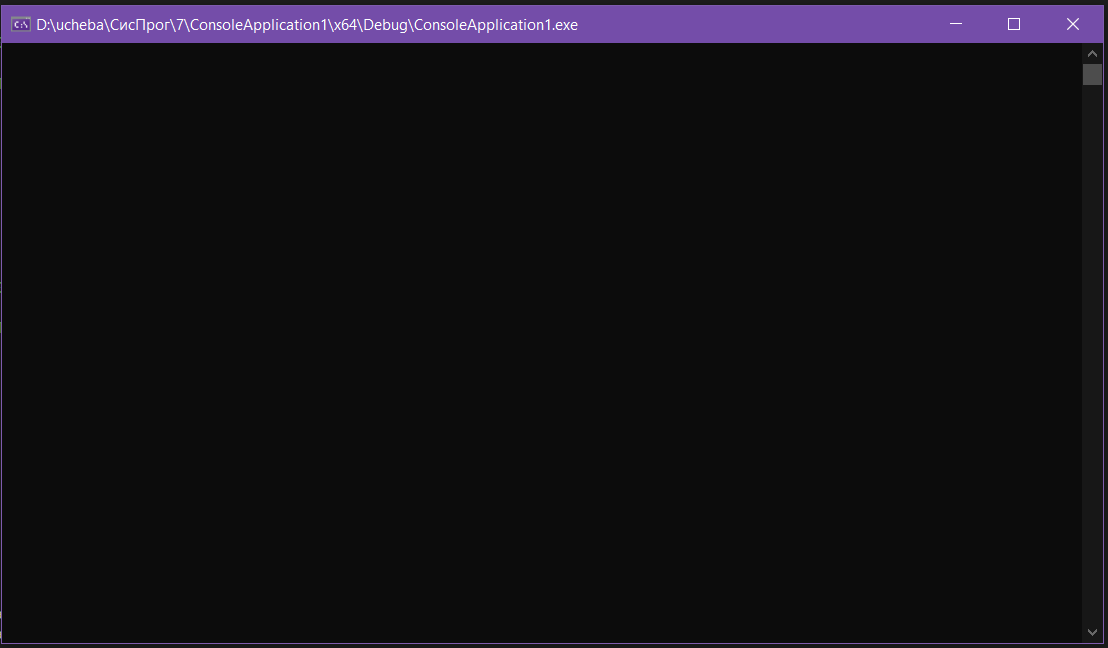
Цели работы:

1. Понять, что такое тупики (deadlocks) в многопоточном программировании.
2. Изучить основные причины возникновения тупиков.
3. Изучить методы предотвращения и устранения тупиков в Windows с использованием C++.
4. Научиться применять функции синхронизации Windows для безопасного завершения тупиков.

Задание 1: Пример тупика

В этом примере два потока захватывают мьютексы в разном порядке, что приводит к тупику.

Результат: программа виснет.



Задание 2: Предотвращение тупика

В этом примере реализуется механизм предотвращения тупиков за счет упорядочивания захвата мьютексов.

Код:

#include <iostream>

#include <windows.h>

#include <io.h>

#include <fcntl.h>

HANDLE mutex1, mutex2; // Два мьютекса

DWORD WINAPI ThreadFunction1(LPVOID lpParam) {

WaitForSingleObject(mutex1, INFINITE); // Захватываем первый мьютекс

Sleep(1000); // Задержка для демонстрации

WaitForSingleObject(mutex2, INFINITE); // Захватываем второй мьютекс

// Освобождаем оба мьютекса

ReleaseMutex(mutex1);

ReleaseMutex(mutex2);

return 0;

}

DWORD WINAPI ThreadFunction2(LPVOID lpParam) {

WaitForSingleObject(mutex1, INFINITE); // Также захватываем первый мьютекс

Sleep(1000); // Задержка для демонстрации

WaitForSingleObject(mutex2, INFINITE); // Захватываем второй мьютекс

// Освобождаем оба мьютекса

ReleaseMutex(mutex1);

ReleaseMutex(mutex2);

return 0;

}

int main() {

// Настройка консоли для вывода Unicode

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

// Создаем мьютексы

mutex1 = CreateMutex(NULL, FALSE, NULL);

mutex2 = CreateMutex(NULL, FALSE, NULL);

HANDLE thread1 = CreateThread(NULL, 0, ThreadFunction1, NULL, 0, NULL);

HANDLE thread2 = CreateThread(NULL, 0, ThreadFunction2, NULL, 0, NULL);

// Ожидаем завершения потоков

WaitForSingleObject(thread1, INFINITE);

WaitForSingleObject(thread2, INFINITE);

// Освобождаем ресурсы

CloseHandle(mutex1);

CloseHandle(mutex2);

CloseHandle(thread1);

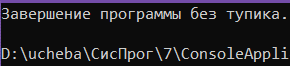
CloseHandle(thread2);

std::wcout << L"Завершение программы без тупика.\n";

return 0;

}

Результат:



Самостоятельные задания:

Модифицируйте код так, чтобы использовать не два, а три мьютекса. Объясните, как изменится вероятность возникновения тупика.

Код:

#include <iostream>

#include <windows.h>

#include <io.h>

#include <fcntl.h>

HANDLE mutex1, mutex2, mutex3; // Три мьютекса

DWORD WINAPI ThreadFunction1(LPVOID lpParam) {

WaitForSingleObject(mutex1, INFINITE); // Захватываем первый мьютекс

Sleep(1000); // Задержка для демонстрации

WaitForSingleObject(mutex2, INFINITE); // Захватываем второй мьютекс

Sleep(1000); // Задержка для демонстрации

WaitForSingleObject(mutex3, INFINITE); // Захватываем третий мьютекс

// Освобождаем все мьютексы

ReleaseMutex(mutex1);

ReleaseMutex(mutex2);

ReleaseMutex(mutex3);

return 0;

}

DWORD WINAPI ThreadFunction2(LPVOID lpParam) {

WaitForSingleObject(mutex1, INFINITE); // Также захватываем первый мьютекс

Sleep(1000); // Задержка для демонстрации

WaitForSingleObject(mutex3, INFINITE); // Захватываем третий мьютекс

Sleep(1000); // Задержка для демонстрации

WaitForSingleObject(mutex2, INFINITE); // Пытаемся захватить второй мьютекс

// Освобождаем все мьютексы

ReleaseMutex(mutex1);

ReleaseMutex(mutex2);

ReleaseMutex(mutex3);

return 0;

}

int main() {

// Настройка консоли для вывода Unicode

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

// Создаем мьютексы

mutex1 = CreateMutex(NULL, FALSE, NULL);

mutex2 = CreateMutex(NULL, FALSE, NULL);

mutex3 = CreateMutex(NULL, FALSE, NULL);

HANDLE thread1 = CreateThread(NULL, 0, ThreadFunction1, NULL, 0, NULL);

HANDLE thread2 = CreateThread(NULL, 0, ThreadFunction2, NULL, 0, NULL);

// Ожидаем завершения потоков

WaitForSingleObject(thread1, INFINITE);

WaitForSingleObject(thread2, INFINITE);

// Освобождаем ресурсы

CloseHandle(mutex1);

CloseHandle(mutex2);

CloseHandle(mutex3);

CloseHandle(thread1);

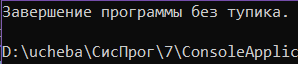
CloseHandle(thread2);

std::wcout << L"Завершение программы без тупика.\n";

return 0;

}

Результат:



Увеличение количества мьютексов: Добавление третьего мьютекса увеличивает сложность взаимодействия между потоками. Теперь вероятность возникновения тупика зависит не только от порядка захвата мьютексов, но и от взаимодействия между всеми тремя мьютексами.

Порядок захвата: В данном коде потоки захватывают мьютексы в разном порядке:

* Первый поток захватывает mutex1, затем mutex2, и наконец mutex3.
* Второй поток захватывает mutex1, затем mutex3, и в конце mutex2.

Из-за этого, если один поток захватит mutex1 и mutex3, а второй поток захватит mutex1 и будет пытаться захватить mutex2, то оба потока могут застрять, ожидая освобождения мьютексов друг у друга, что и приведет к тупику.

**Контрольные вопросы:**

1. Что такое тупик?

Тупик (Deadlock) — это состояние в многозадачной системе, при котором два или более процесса (или потока) блокируют друг друга, ожидая освобождения ресурсов, удерживаемых другим процессом. В результате каждый процесс находится в состоянии ожидания, и выполнение программы не может продолжиться.

1. Какие четыре условия необходимы для возникновения тупика?

Основные условия, при которых может возникнуть тупик:

* Взаимное исключение (Mutual Exclusion): хотя бы один ресурс находится в режиме эксклюзивного доступа и не может быть использован другим процессом.
* Удержание и ожидание (Hold and Wait): процесс, который уже удерживает один ресурс, может запросить дополнительные ресурсы.
* Отсутствие принудительного освобождения (No Preemption): ресурсы не могут быть принудительно отобраны у процессов, их можно только добровольно освободить.
* Циклическое ожидание (Circular Wait): существует замкнутый цикл процессов, где каждый процесс ожидает ресурса, удерживаемого следующим процессом в цепочке.

1. Чем отличается взаимное блокирование от циклического ожидания?

При взаимном блокировании один ресурс, необходимый одному процессу, удерживается другим, а циклическое ожидание – это ситуация в которой цикл процессов ожидает ресурс, удерживаемый другим процессом.

1. Как можно предотвратить возникновение тупиков?

Чтобы избежать тупиков, используются различные методы:

* Избегание тупиков (Deadlock Avoidance): например, алгоритм банкара, который проверяет, будет ли система в тупике, если выделить ресурсы процессу.
* Предотвращение тупиков (Deadlock Prevention): один из методов — захват ресурсов в одном и том же порядке всеми потоками, как это реализовано в нашем примере.
* Обнаружение и разрешение тупиков (Deadlock Detection and Resolution): в этом случае система периодически проверяет наличие тупиков и предпринимает шаги для их устранения.

1. Какой порядок захвата ресурсов поможет избежать тупика?

Захват ресурсов в одном и том же порядке всеми потоками. Это предотвращает циклическое ожидание, так как все потоки будут следовать одному и тому же порядку при захвате ресурсов.

1. Какие функции Windows используются для создания и захвата мьютексов?

CreateMutex: для создания мьютекса.

WaitForSingleObject: для захвата мьютекса.

1. Какие методы борьбы с тупиками существуют в Windows?

График распределения ресурсов: Windows использует алгоритм на основе графа для обнаружения потенциальных взаимоблокировок. Он использует график распределения ресурсов для определения циклов, указывающих на потенциальную взаимоблокировку. После обнаружения взаимоблокировки Windows предпринимает соответствующие действия для ее устранения и позволяет процессам продолжить работу.

1. Как можно обнаружить тупик в многопоточной системе?

Одним из способов обнаружения тупика является использование графа ресурсов, где узлы представляют процессы и ресурсы. Если в графе существует цикл, это указывает на наличие тупика. Также можно использовать тайм-ауты при ожидании ресурсов и проверять состояние потоков.

1. Чем различаются мьютексы и критические секции в контексте борьбы с тупиками?

Мьютексы: могут работать между процессами и обеспечивают механизм синхронизации для потоков, работающих в разных процессах. Мьютексы могут быть захвачены и освобождены в любом порядке, что может привести к тупикам.

Критические секции: используются для синхронизации потоков внутри одного процесса. Они работают быстрее, чем мьютексы, но не могут быть использованы между процессами. Критические секции также могут быть организованы так, чтобы избежать тупиков, если все потоки захватывают их в одном порядке.