**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.

**Теоретический материал:**

**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. Тупики возникают в системах, где используются **мьютексы**, **семафоры** и другие механизмы синхронизации для управления доступом к ресурсам.

Основные условия, при которых может возникнуть тупик:

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

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.

**Задание 1: Пример тупика**

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

**Код:**

#include <iostream>

#include <windows.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); // Пытаемся захватить второй мьютекс

// Освобождаем оба мьютекса

ReleaseMutex(mutex1);

ReleaseMutex(mutex2);

return 0;

}

DWORD WINAPI ThreadFunction2(LPVOID lpParam) {

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

Sleep(1000); // Задержка для демонстрации

WaitForSingleObject(mutex1, INFINITE); // Пытаемся захватить первый мьютекс

// Освобождаем оба мьютекса

ReleaseMutex(mutex2);

ReleaseMutex(mutex1);

return 0;

}

int main() {

// 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::cout << "Завершение программы.\n";

return 0;

}

**Ожидаемый результат:**

Программа "зависнет", так как два потока будут ожидать освобождения ресурсов друг у друга, и это приведет к тупику.

**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;

}

**Ожидаемый результат:**

Программа завершится корректно, поскольку оба потока захватывают мьютексы в одном и том же порядке, что предотвращает тупик.

**Самостоятельные задания:**

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

**Контрольные вопросы:**

1. Что такое тупик?
2. Какие четыре условия необходимы для возникновения тупика?
3. Чем отличается взаимное блокирование от циклического ожидания?
4. Как можно предотвратить возникновение тупиков?
5. Какой порядок захвата ресурсов поможет избежать тупика?
6. Какие функции Windows используются для создания и захвата мьютексов?
7. Какие методы борьбы с тупиками существуют в Windows?
8. Как можно обнаружить тупик в многопоточной системе?
9. Чем различаются мьютексы и критические секции в контексте борьбы с тупиками?