# Лабораторная работа № 3. MUTEX

## Функции для работы с мьютексами

**Мьютекс** *(mutex - mutual exclusion - “взаимное исключение”)* – системный объект операционной системы, предназначенный для синхронизации процессов и/или потоков. Мьютекс может находится в двух состояниях – открыт и закрыт. Если мьютекс находится в состоянии «открыт», то поток может захватить его и, следовательно, войти в свою критическую секцию. При захвате мьютекс переходит в состояние «закрыт». По окончании выполнения критической секции поток обязан освободить мьютекс. Если поток пытается захватить мьютекс, находящийся в состоянии «закрыт», то он приостанавливается и ждет освобождения мьютекса.

В операционной системе Microsoft Windows мьютексы реализованы как объекты ядра системы, а поэтому могут быть использованы для синхронизации потоков, запущенных в контексте разных процессов. Доступ к мьютексу осуществляется через его handle.

Для создания мьютекса используется функция **CreateMutex**:

HANDLE CreateMutex(

LPSECURITY\_ATTRIBUTES lpMutexAttributes,

BOOL bInitialOwner,

LPCTSTR lpName

);

Значения атрибутов функции CreateMutex:

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| lpMutexAttributes | Атрибуты безопасности мьютекса. Если данный параметр равен NULL, то мьютекс получает дескриптор безопасности, установленный по умолчанию, а указатель мьютекса, полученный в результате вызова CreateMutex не может быть унаследован дочерним процессом. |
| bInitialOwner | Устанавливает владельца мьютекса. Если данный параметр равен true, то владельцем является поток, создавший мьютекс. Мьютекс создается в закрытом состоянии. В противном случае, если данный параметр равен false, поток владельцем мьютекса не является, и мьютекс создается в состоянии «открыт». |
| lpName | Имя мьютекса. Если данный параметр не равен NULL, то мьютексу присваивается имя, используя которое другие потоки могут открыть и использовать данный мьютекс. Мьютекс с установленным именем называется поименованным. Если данный параметр равен NULL, то мьютекс считается анонимным, и его могут использовать только потоки данного процесса и потоки потомков данного процесса, которые унаследовали указатель мьютекса (при наличии соответствующего дескриптора безопасности). |

Возвращаемые значения функции CreateMutex:

1. Если данная функция вернула значение NULL, то мьютекс не был создан. Дополнительную информацию об ошибке можно получить, используя функцию GetLastError.

2. В противном случае функция возвращает указатель мьютекса (handle).

3. Если при создании поименованного мьютекса оказывается, что такой мьютекс уже существует, то функция CreateMutex все равно возвращает его дескриптор (при этом объект не пересоздается), а функция GetLastError возвращает значение ERROR\_ALREADY\_EXISTS.

Открытие существующего мьютекса можно выполнить функцией **OpenMutex**:

HANDLE OpenMutex(

DWORD dwDesiredAccess,

BOOL bInheritHandle,

LPCTSTR lpName);

Параметры функции OpenMutex:

Второй и третий параметры данной функции аналогичны соответствующим параметрами функции CreateMutex. Первый параметр (dwDesiredAccess) определяет параметры доступа к объекту мьютекса из процесса, открывшего данный мьютекс. Параметр может принимать 2 значения:

|  |  |
| --- | --- |
| **Значение** | **Комментарий** |
| MUTEX\_ALL\_ACCESS | Процесс, выполнивший открытие мьютекса, получает все возможные права доступа. |
| MUTEX\_MODIFY\_STATE | Процесс получает права только на изменение состояния мьютекса, которые нужны для выполнения функции ReleaseMutex |

Возвращаемые значения функции OpenMutex:

1. Если открытие мьютекса прошло успешно, и поток получил доступ к мьютексу, то функция возвращает указатель (handle) мьютекса.

2. Если в процессе открытия мьютекса произошла ошибка, то функция вернет значение NULL.

Дождаться освобождения мьютекса и захватить его перед входом в критическую секцию можно при помощи функции **WaitForSingleObject**:

DWORD WaitForSingleObject(HANDLE hHandle, DWORD dwMilliseconds);

Если же потоку необходимо дождаться освобождения и захватить несколько мьютексов (или один из нескольких мьютексов), то можно воспользоваться функцией **WaitForMultipleObjects**:

DWORD WaitForMultipleObjects(

DWORD nCount,

const HANDLE\* lpHandles,

BOOL bWaitAll,

DWORD dwMilliseconds);

Значение параметров данных функций аналогично параметрам при работе с потоками за исключением того, что в качестве дескриптора hHandle используется дескриптор мьютекса, а вместо массива дескрипторов потоков lpHandles используется массив дескрипторов мьютексов.

При работе с мьютексами данные функции выполняют проверку состояния мьютекса:

* если мьютекс открыт, то перевести его в состояние «закрыт» и войти в критическую секцию;
* в противном случае – приостановить выполнение потока до того момента, пока мьютекс не окажется в состоянии «открыт».

После прохождения потоком барьера WaitForSingleObject или WaitForMultipleObjects, мьютексы, проверяемые потоком, оказываются закрытыми. Поэтому после окончания критической секции их необходимо открыть функцией **ReleaseMutex**:

BOOL ReleaseMutex(HANDLE hMutex);

В качестве параметра функции передается указатель (handle) открываемого мьютекса. Если мьютекс удалось перевести в состояние «открыт» функция вернет значение true, в противном случае – значение false.

В конце программы необходимо отключить все мьютексы при помощи функции **CloseHandle**.

## Пример программы

***Задача***. В качестве примера программы рассмотрим классическую проблему синхронизации «Производитель-Потребитель». В данной задаче существует два потока (процесса) один, из которых является производителем информации, а другой – ее потребителем. На их работу наложены следующие условия:

- производитель не может сгенерировать новую порцию информации, пока потребитель не обработал имеющуюся;

- потребитель может выполнять свою работу только в том случае, когда производитель сгенерировал очередную порцию информации.

В качестве частного примера такой задачи рассмотрим такой вариант:

1. Производитель генерирует простые выражения вида

<целое число><операция><целое число>,

где <операция> - это арифметическая операция сложения (+), вычитания (−), умножения (\*) или деления (/).

Выражения могут быть введены пользователем.

1. Потребитель обязан разобрать выражение и вывести результаты на экран.

***Решение.*** Для решения данной задачи необходимо ввести два потока – производитель и потребитель, и два мьютекса – «информация сгенерирована» и «информация обработана».

Производитель может начать работу, только если мьютекс «информация обработана» открыт. При этом он должен захватить мьютекс «информация сгенерирована». После генерации информации он должен освободить мьютекс «информация сгенерирована».

Потребитель поступает наоборот – начинает свою работу с открытым мьютексом «информация сгенерирована», а затем манипулирует мьютексом «информация обработана».

#include <windows.h>

#include <stdio.h>

#include <iostream>

HANDLE hMutexReady, // Мьютекс "информация сгенерирована"

hMutexHandled, // Мьютекс "информация обработана"

hMutex; // Мьютекс доступа к переменной

char expression[100]; // разделяемая переменная

// Функция потока производителя

DWORD WINAPI producer(LPVOID param)

{

int a, b;

char oper;

while (true)

{

// Захватить мьютекс "информация обработана"

WaitForSingleObject(hMutexHandled, INFINITE);

// Обратиться к критической секции и сгенерировать выражение

WaitForSingleObject(hMutex,INFINITE);

std::cout << "Введите выражение";

std::cin >> a >> oper >> b;

sprintf(expression, "%d%c%d", a, oper, b);

// Освободить критическую секцию

ReleaseMutex(hMutex);

// Освободить мьютекс "информация обработана"

ReleaseMutex(hMutexReady);

}

return 0;

}

// Функция потока потребителя

DWORD WINAPI consumer(LPVOID param)

{

int a, b, res;

char oper;

while (true)

{

// Захватить мьютекс "информация сгенерирована"

WaitForSingleObject(hMutexReady,INFINITE);

// Обратиться к критической секции и вычислить значение выражения

WaitForSingleObject(hMutex,INFINITE);

sscanf(expression, "%d%c%d", &a, &oper, &b);

switch(oper)

{

case '+': res = a + b; break;

case '-': res = a - b; break;

case '\*': res = a \* b; break;

case '/': res = (b?a/b:0);break;

default: res = 0;

}

std::cout << "\nSource operation \n"

<< a << oper << b << '\n';

if (oper=='+' || oper=='-' || oper=='\*')

std::cout << "Res = " << res << '\n';

else if (oper == '/' && res == 0 && b != 0)

std::cout << "Res = " << res << '\n';

else std::cout << "Error \n";

std::cout << '\n';

// Освободить критическую секцию

ReleaseMutex(hMutex);

// Освободить мьютекс "информация обработана"

ReleaseMutex(hMutexHandled);

}

return 0;

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

// Создаем мьютексы со значениями по умолчанию

// все мьютексы созданы в состоянии "открыт"

hMutexReady = CreateMutex(NULL, false, NULL);

hMutexHandled = CreateMutex(NULL, false, NULL);

hMutex = CreateMutex(NULL, false, NULL);

// Создаем потоки производителя и потребителя

HANDLE hProducer = CreateThread(NULL,0,producer,NULL,0,NULL);

HANDLE hConsumer = CreateThread(NULL,0,consumer,NULL,0,NULL);

HANDLE hThread[2];

hThread[0] = hProducer;

hThread[1] = hConsumer;

// Дожидаемся завершения обоих потоков

WaitForMultipleObjects(2, hThread, true, INFINITE);

CloseHandle(hProducer);

CloseHandle(hConsumer);

return 0;

}

## Задания

1. Написать программу для нахождения произведения векторов, значения которых вводятся пользователем. Каждое скалярное произведение должно обрабатываться отдельным потоком.
2. Написать 2 консольных приложения. Первое приложение должно запрашивать у пользователя текст и сохранять их в текстовый файл, а второе - читать текст из текстового файла и выводить на экран. Каждая программа должна показывать своё текущее состояние (например, программа чтения из файла должна отображать состояние ожидания открытия мьютекса).
3. Написать программу распределения чисел в хеш-таблицу. Имеется n потоков, генерирующих случайные целые числа. Для определения местоположения числа x в хеш-таблице использовать следующую хеш-функцию:

int hash(int x)

{

return x % k;

}

Если поток собирается записать число x в i-ю строку таблицы, то он должен проверить, не пишет ли другой поток свое число в эту же строку.

Количество строк таблицы и основание хеш-функции (число k) определяется пользователем.

Операции работы с хеш-таблицей вынести в отдельный класс (реализовать модель монитора), например:

class CMonitor

{

int \* occupated\_rows;

…

public:

void OcupateRow(int row\_number);

void FreeRow(int row\_number);

…

}

Определите и используйте необходимое количество мьютексов.