# Лабораторная работа № 1. Многопоточные приложения в MS Windows

## Основные функции для работы с потоками

Для создания нового потока используется функция **CreateThread**:

HANDLE CreateThread(

LPSECURITY\_ATTRIBUTES lpThreadAttributes,

SIZE\_T dwStackSize,

LPTHREAD\_START\_ROUTINE lpStartAddress,

LPVOID lpParameter,

DWORD dwCreationFlags,

LPDWORD lpThreadId

);

Параметры функции **CreateThread**:

|  |  |
| --- | --- |
| *Параметр* | *Описание* |
| lpThreadAttributes | Указатель на структуру типа SECURITY\_ATTRIBUTES, в которой указывается, может ли возвращенный функцией CreateThread указатель на поток (HANDLE) быть унаследованным дочерним процессом данного процесса. Если данный параметр равен NULL, то указатель не может быть унаследован. |
| dwStackSize | Длина стека потока в байтах. Система округляет это значение до размера ближайшей страницы. Если значение этого параметра равно 0, то используется размер потока по умолчанию, который определяется по максимально допустимому размеру виртуального адресного пространства процесса. |
| lpStartAddress | Определяет имя функции, которая должна быть выполнения в потоке. Функция для выполнения в потоке описывается специальным образом (см. ниже). |
| lpParameter | Нетипизированный указатель на структуру, которая будет передана в качестве параметра функции потока. Т.к. функции потока могут иметь только один параметр типа LPVOID, то все параметры, которые должны быть переданы в поток, должны быть сведены в одну структуру. |
| dwCreationFlags | Набор флагов, влияющих на создание потока. При указании значения 0 поток запускается непосредственно после создания. |
| lpThreadId | Указатель на переменную типа DWORD, в которую возвращается идентификатор потока. Если данный параметр равен NULL, то идентификатор не возвращается. |

Возвращаемые значения функции **CreateThread**:

1. Если функция выполнилась успешно, то она вернет значение HANDLE для созданного потока.

2. Если во время выполнения функции произошла ошибка, то функция вернет значение NULL. При необходимости можно воспользоваться функцией GetLastError для получения большей информации о возникшей ошибке.

Дополнительные сведения

1. Функция, которая может быть запущена в потоке, должна быть описана следующим образом:

DWORD WINAPI <имя функции> (LPVOID <имя параметра>)

{

<последовательность операторов>;

}

2. Создание потока со значением NULL в качестве атрибутов безопасности потока, означает, что поток будет создан с атрибутом THREAD\_ALL\_ACCESS, т.е. Windows предоставит потоку все возможные права доступа. В случае необходимости программист может изменить права потока, создав структуру параметров безопасности.

3. При создании потока программист может использовать следующие флаги создания:

|  |  |
| --- | --- |
| *Флаг создания (значение)* | *Значение* |
| 0 | Поток будет запущен непосредственно после создания |
| CREATE\_SUSPENDED  0x00000004 | Поток будет создан в приостановленном состоянии. Для запуска потока необходимо использовать функцию ResumeThread, передав ей в качестве параметра HANDLE потока. |
| STACK\_SIZE\_PARAM\_IS\_A\_RESERVATION  0x00010000 | Если этот флаг указан, то Windows обязательно создаст стек потока с указанном в параметре <размер стека потока> размером. В противном случае будет выбран размер стека, принятый в операционной системе. |

Для закрытия потока используется функция **CloseHandle**:

BOOL CloseHandle(

HANDLE hHandle

);

В качестве параметра функция получает указатель (handle) потока. Если закрытие потока происходит нормально, то функция возвращает не нулевое значение, или true. В противном случае функция возвращает значение 0, или false. Функция **CloseHandle** также используется для закрытия других объектов ядра, использованных в программе пользователя – мьютексов, семафоров и т.д.

После запуска потока (потоков) бывает необходимо дождаться их завершения. Для этого используются две функции ожидания.

Ожидание завершения одного потока может быть выполнено функцией **WaitForSingleObject**:

DWORD WaitForSingleObject(

HANDLE hHandle,

DWORD dwMilliseconds

);

Параметры функции **WaitForSingleObject**:

|  |  |
| --- | --- |
| *Параметр* | *Описание* |
| hHandle | Указатель ожидаемого потока |
| dwMilliseconds | Время ожидания завершения в миллисекундах. Если время ожидания точно установить не удается, а программе необходимо дождаться полного выполнения потока, то в качестве этого параметра можно использовать значение INFINE (бесконечное ожидание) |

Возвращаемые значения функции **WaitForSingleObject**:

WAIT\_OBJECT\_0 – ожидание завершения потока прошло успешно;

WAIT\_TIMEOUT – поток не успел завершиться в течение указанного времени.

Ожидание завершения нескольких потоков может быть выполнено функцией **WaitForMultipleObjects**:

DWORD WaitForMultipleObjects(

DWORD nCount,

const HANDLE\* lpHandles,

BOOL bWaitAll,

DWORD dwMilliseconds

);

Параметры функции **WaitForMultipleObjects**:

|  |  |
| --- | --- |
| *Параметр* | *Описание* |
| nCount | Количество потоков, завершения которых необходимо дождаться |
| lpHandle | Указатель на массив указателей потоков |
| bWaitAll | Флаг ожидания, который показывает, нужно ли дожидаться завершения всех потоков. Если значение этого флага true, тогда поток, остановившийся на функции WaitForMultipleObjects, продолжит свою работу только после завершения всех ожидаемых потоков. При значении флага, равного false, достаточно завершения хотя бы одного из ожидаемых потоков. |
| dwMilliseconds | Время ожидания завершения в миллисекундах. Если время ожидания точно установить не удается, а программе необходимо дождаться полного выполнения потока, то в качестве этого параметра можно использовать значение INFINE (бесконечное ожидание) |

Надо отметить, что функции **WaitForSingleObject** и **WaitForMultipleObjects** используются при работе и с другими системными объектами, например, для ожидания освобождения семафора или мьютекса.

Также для работы с потоками могут быть использованы другие функции WinAPI (см. табл. 1)

Таблица 1. Некоторые функции для работы с потоками

|  |  |
| --- | --- |
| *Функция* | *Описание* |
| VOID ExitThread (  DWORD dwExitCode  ); | Функция выхода из потока, в качестве параметра передается код выхода. Является предпочтительной функцией для завершения работы потока |
| HANDLE GetCurrentThread(void); | Возвращает указатель текущего потока |
| DWORD GetCurrentThreadId(void); | Возвращает идентификатор текущего потока |
| DWORD GetThreadId(  HANDLE Thread  ); | Возвращает идентификатор потока по его указателю |
| int GetThreadPriority(  HANDLE hThread  ); | Возвращает приоритет потока по его указателю |
| HANDLE OpenThread(  DWORD dwDesiredAccess,  BOOL bInheritHandle,  DWORD dwThreadId  ); | Функция открытия потока |
| DWORD ResumeThread(  HANDLE hThread  ); | Возобновление работы потока |
| BOOL SetThreadPriority(  HANDLE hThread,  int nPriority  ); | Установить приоритет потока |
| VOID Sleep(  DWORD dwMilliseconds  ); | Приостановить текущий поток на указанный промежуток времени |
| DWORD SuspendThread(  HANDLE hThread  ); | Приостановить указанный поток до его возобновления функцией ResumeThread |

Более подробную информацию по этим и другим функциям для управления потоками можно получить из MSDN Library или на сайте <http://msdn.com>. В настоящем пособии приведем лишь константы, используемые для установки приоритета потока (см. табл. 2)

Таблица 2. Константы приоритета потоков

|  |  |
| --- | --- |
| *Константа приоритета* | *Описание* |
| THREAD\_PRIORITY\_ABOVE\_  NORMAL | Приоритет выше среднего (на 1 больше, чем класс приоритета процесса, запустившего данный поток) |
| THREAD\_PRIORITY\_BELOW\_  NORMAL | Приоритет выше среднего (на 1 меньше, чем класс приоритета процесса, запустившего данный поток) |
| THREAD\_PRIORITY\_HIGHEST | Высокий приоритет (на 2 больше, чем класс приоритета процесса, запустившего данный поток) |
| THREAD\_PRIORITY\_IDLE | Базовый приоритет будет равен 1, если класс приоритета процесса:  IDLE\_PRIORITY\_CLASS (простаивающий), BELOW\_NORMAL\_PRIORITY\_CLASS (ниже среднего),  NORMAL\_PRIORITY\_CLASS (обычный), ABOVE\_NORMAL\_PRIORITY\_CLASS (выше среднего)  HIGH\_PRIORITY\_CLASS (высокий)  Базовый приоритет потока будет равен 16, если приоритет процесса REALTIME\_PRIORITY\_CLASS (процесс реального времени). |
| THREAD\_PRIORITY\_LOWEST | Низкий приоритет (на 2 меньше, чем класс приоритета процесса, запустившего данный поток) |
| THREAD\_PRIORITY\_NORMAL | Обычный приоритет для данного класса приоритета потока |
| THREAD\_PRIORITY\_TIME\_  CRITICAL | Базовый приоритет потока будет равен 15, если класс приоритета процесса:  IDLE\_PRIORITY\_CLASS, BELOW\_NORMAL\_PRIORITY\_CLASS,  NORMAL\_PRIORITY\_CLASS, ABOVE\_NORMAL\_PRIORITY\_CLASS,  HIGH\_PRIORITY\_CLASS  Базовый приоритет потока будет равен 31, если класс приоритета процесса  REALTIME\_PRIORITY\_CLASS |

## Пример программы

***Задача.*** Дана матрица из вещественных чисел, содержащая m строк и n столбцов. Необходимо найти номер столбца матрицы, сумма элементов которого минимальна.

***Решение.*** В принципе, данная задача просто решается и без использования потоков, но мы поступим следующим образом – для подсчета суммы каждого из столбцов запустим собственный поток, а после того, как все потоки завершат свою работу, найдем минимальную сумму и определим, какому из столбцов она принадлежит.

В создаваемые потоки необходимо передавать номера обрабатываемых столбцов. Обратите внимание на то, как это сделано в программе. Передавать указатель на счетчик цикла в данном случае невозможно – если мы передадим указатель на переменную, а потом изменим ее значение, то явно получим неверный результат. Поэтому для передачи параметров создадим дополнительный массив.

#include <windows.h>

#include <iostream>

const int m = 10, n = 20;

float mtx[m][n], // матрица

col\_sums[n]; // массив сумм

int col\_numbers[n]; // номера столбцов

// Функция потока

DWORD WINAPI col\_sum (LPVOID param)

{

// Получаем значение параметра - номер столбца

int \* pcol\_num = (int \*) param;

int col\_num = \*pcol\_num;

// Находим искомую сумму

col\_sums[col\_num] = 0;

for (int i = 0; i<m; i++)

{

col\_sums[col\_num] += (mtx[i][col\_num]);

}

return 0;

}

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

{

// Описание переменных для работы с потоками

HANDLE hThread[n]; // -- массив из n указателей потоков

DWORD dwThreadID[n]; // -- массив из n идентификаторов потоков

// Создание матрицы

for (int i = 0; i<m; i++)

{

for (int j = 0; j<n; j++)

{

mtx[i][j] = (float)(rand());

}

}

// Заполнение массивов исходными значениями

for (int i = 0; i<n; i++)

{

col\_sums[i] = 0;

col\_numbers[i] = i;

}

//Запуск потоков

for (int i = 0; i<n; i++)

{

hThread[i] = CreateThread(

// атрибуты безопасности по умолчанию

NULL,

// размер стека по умолчанию

0,

// имя функции

col\_sum,

// указатель на параметры

&(col\_numbers[i]),

// флаг создания = 0

0,

// адрес переменной для идентификатора &(dwThreadID[i]));

// Проверям - создан ли поток

if (hThread[i] == NULL)

{

std::cout << "Поток № " << i << "не был создан\n" << "Ошибка " << GetLastError();

}

}

// Ожидаем завершения потоков

WaitForMultipleObjects(n, hThread, true, INFINITE);

// Находим номер столбца с минимальной суммой

int num\_min = 0;

float min = col\_sums[0];

for (int i = 1; i<n; i++)

{

if (min > col\_sums[i])

{

min = col\_sums[i];

num\_min = i;

}

}

// Вывод результата

std::cout << "Искомый столбец № " << num\_min << '\n';

// Закрытие потоков

for (int i = 0; i<n; i++)

{

CloseHandle(hThread[i]);

}

return 0;

}

## Задания

1. Измените приведенную в примере программу так, чтобы создание строк матрицы происходило параллельно.

*Далее во всех заданиях с матрицами использовать паралелльное создание строк.*

1. Дана матрица из вещественных чисел, содержащая n строк и m столбцов. Отсортируйте каждую строку матрицы по убыванию методом подсчёта.
2. Дана матрица из натуральных чисел, содержащая n строк и m столбцов. Найдите результат умножения матрицы на множитель, вводимый пользователем.
3. Напишите программу, выводящую на форму круги трёх разных цветов в случайных местах (три разных потока). Предусмотрите возможность изменить приоритет потоков управляющими элементами на форме.
4. Дано бинарное дерево, элементы которого содержат целые числа. Найдите сумму элементов этого дерева и среднее значение.
5. Дан текст длиной n символов и ключ длиной k символов. Осуществите блочное шифрование текста по следующему алгоритму:

* Разделить текст на блоки длиной k символов. Если n не кратно k, то допустимо, чтобы длина последнего блока была меньше k.
* Внутри каждого блока выполнить перестановку символов так, чтобы первый символ занял место последнего, второй – предпоследнего и т.д. Последний символ должен оказаться на месте первого символа блока.
* Применить ключ к каждому блоку. Шифрованный i-й символ блока должен быть получен, как результат исключающего или между i-м исходным символом блока и i-м символом ключа.

1. Напишите программу, выполняющую дешифрование текста, полученного в результате применения описанного алгоритма при известном ключе.