Потоки в Windows

- *Потоком в Windows* называется объект ядра, которому операционная система выделяет процессорное время для выполнения приложения.
- Каждому потоку принадлежат следующие ресурсы:

- код исполняемой функции;
- набор регистров процессора;
- стек для работы приложения;
- стек для работы операционной системы;
- маркер доступа, который содержит информацию для системы безопасности.

- В операционных системах Windows различаются потоки двух типов:

✔ системные потоки;

✔ пользовательские потоки.

- В работающем приложении различаются потоки двух типов:

✔ рабочие потоки (working threads);

✔ потоки интерфейса пользователя (user interface threads).

- Создается поток функцией **createThread,** которая имеет следующий прототип:

```
HANDLE CreateThread(
LPSECURITY_ATTRIBUTES lpThreadAttributes, // атрибуты защиты
DWORD dwStackSize,   // размер стека потока в байтах
LPTHREAD_START_ROUTINE lpStartAddress,   // адрес функции
LPVOID lpParameter      // адрес параметра
DWORD dwCreationFlags, // флаги создания потока
LPDWORD lpThreadId   // идентификатор потока
);
```

```
#include <windows.h>
#include <iostream.h>

volatile int n;

DWORD WINAPI Add(LPVOID iNum)
{
  cout << "Thread is started." << endl;
  n += (int)iNum;
  cout << "Thread is finished." << endl;

  return 0;
}
```

```
int main()
{
  int  inc = 10;
  HANDLE  hThread;
  DWORD  IDThread;

  cout << "n = " << n << endl;
  // запускаем поток Add
  hThread = CreateThread(NULL, 0, Add, (void*)inc, 0, &IDThread);
  if (hThread == NULL)
    return GetLastError();

  // ждем, пока поток Add закончит работу
  WaitForSingleObject(hThread, INFINITE);
  // закрываем дескриптор потока Add
  CloseHandle(hThread);

  cout << "n = " << n << endl;

  return 0;
```

- Поток завершается вызовом функции **ExitThread,** которая имеет следующий прототип:

```
VOID ExitThread(
DWORD dwExitCode   // код завершения потока
);
```

- Один поток может завершить другой поток, вызвав функцию **TerminateThread,** которая имеет следующий прототип:

```
BOOL TerminateThread(
HANDLE hThread,  // дескриптор потока
DWORD dwExitThread // код завершения потока
);
```

- Исполнение каждого потока может быть приостановлено вызовом функции **suspendThread,** которая имеет следующий прототип:

```
DWORD SuspendThread(
HANDLE hThread      // дескриптор потока
);
```

- Эта функция увеличивает значение счетчика приостановок на 1 и, при успешном завершении, возвращает текущее значение этого счетчика. В случае неудачи функция **SuspendThread** возвращает значение, равное -1.

- Для возобновления исполнения потока используется функция **ResumeThread,** которая имеет следующий прототип:

DWORD ResumeThread (

HANDLE hThread // дескриптор потока

- Функция **ResumeThread** уменьшает значение счетчика приостановок на 1 при условии, что это значение было больше нуля. Если полученное значение счетчика приостановок равно 0, то исполнение потока возобновляется, в противном случае поток остается в подвешенном состоянии.

- Поток может задержать свое исполнение вызовом функции sleep, которая имеет следующий прототип:

```
VOID Sleep(
DWORD dwMilliseconds // миллисекунды
);
```

```
#include <windows.h>
#include <iostream.h>

volatile UINT  nCount;
volatile DWORD  dwCount;

void thread()
{
  for (;;)
  {
    nCount++;
    // приостанавливаем поток на 100 миллисекунд
    Sleep(100);
```

```
}

int main()
{
  HANDLE   hThread;
  DWORD  IDThread;
  char c;

  hThread = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)thread, NULL,
                         0, &IDThread);
  if (hThread == NULL)
    return GetLastError();

  for (;;)
  {
    cout << "Input :" << endl;
    cout << "\t'n' to exit" << endl;
    cout << "\t'y' to display the count" << endl;
    cout << "\t's' to suspend thread" << endl;
    cout << "\t'r' to resume thread" << endl;
    cin >> c;
```

```
if (c == 'n')
  break;
switch (c)
{
case 'y':
  cout << "count = " << nCount << endl;
  break;
case 's':
  // приостанавливаем поток thread
  dwCount = SuspendThread(hThread);
  cout << "Thread suspend count = " << dwCount << endl;
  break;
case 'r':
  // возобновляем поток thread
  dwCount = ResumeThread(hThread);
  cout << "Thread suspend count = " << dwCount << endl;
  break;
```

```
    }
  }

  // прерываем выполнение потока thread
  TerminateThread(hThread, 0);
  // закрываем дескриптор потока thread
  CloseHandle(hThread);

  return 0;
}
```