Управление потоками в WinAPI

МДК 01.01 «Системное программирование»

Особенности потоков в WinAPI

Потоком в WinAPI называется объект ядра(или последовательность инструкций программы), которому ОС выделяет процессорное время для выполнения приложения

Потоку принадлежат ресурсы:

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

Все эти ресурсы (или содержимое памяти) образуют **контекст потока** в WinAPI.

Поток имеет **дескриптор** и **идентификатор**, который уникален для потоков выполняющихся в системе. Идентификаторы потоков используются служебными программами, которые позволяют пользователям системы отслеживать работу потоков.

Особенности потоков в WinAPI

В ОС Windows различаются потоки двух типов:

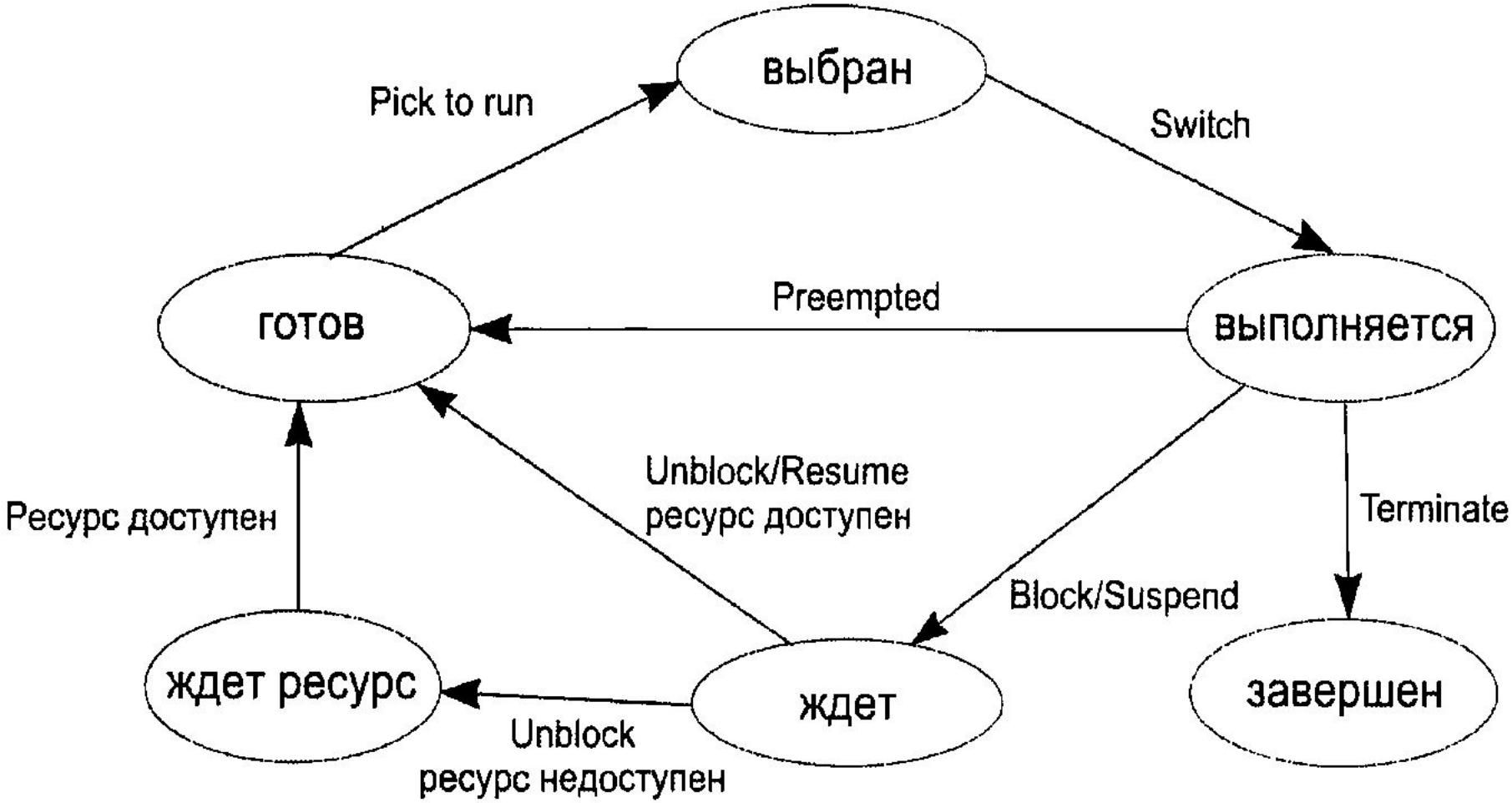
- **системные** потоки – выполняют различные сервисы операционной системы и запускаются ядром операционной системы;
- **пользовательские** потоки – служат для решения задач пользователя и запускаются приложением.

В работающем приложении имеются потоки двух типов:

- **рабочие потоки** (**working threads**) – выполняют различные фоновые задачи в приложении;
- **потоки интерфейса пользователя** (**user interface threads**) – связаны с окнами и выполняют обработку сообщений, поступающих этим окнам.

Каждое приложение имеет, по крайней мере, один поток, который называется **первичным** (**primary**) или **главным** (**main**) потоком - поток, который исполняет функцию ***main*** или ***WinMain***.

Диаграмма состояний потока, работающего в Windows



Создание потоков

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  )
```

При успешном завершении возвращает дескриптор созданного потока и его идентификатор, который является уникальным для всей системы. Иначе возвращает **NULL**.

Создание потоков

lpThreadAttributes – атрибуты защиты потока (= **NULL** – ОС сама установит атрибуты защиты потока по умолчанию).

dwStackSize – размер стека, который выделяется потоку при запуске (= 0 - потоку выделяется минимальный стек 1 Мбайт).

lpStartAddress – указывает на исполняемую потоком функцию:

```
DWORD WINAPI имя_функции_потока(LPVOID lpParameters);
```

lpParameter – указатель на пустой тип.

dwCreationFlags – определяет, в каком состоянии будет создан поток (= 0 – функция потока начинает выполняться сразу после создания потока, = **CREATE_SUSPENDED** – поток создается в подвешенном состоянии, может быть запущен функцией **ResumeThread**).

Создание потоков

`lpThreadId` – выходной, его значение устанавливает Windows, указывает на переменную, в которую Windows поместит уникальный идентификатор потока:

- Идентификатор потока уникален для системы
- Используется для ссылок на поток системными функциями, редко – функциями приложения
- Действителен только на время существования потока.
- После завершения потока тот же идентификатор может быть присвоен другому потоку.

Пример 1. Создание потока функцией CreateThread

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

using namespace std;

volatile int n;
// volatile - указание компилятору
// хранить значение переменной n
// в памяти, а не в регистре

DWORD WINAPI Add(LPVOID iNum)
// Функция, вызываемая
// в отдельном потоке
{
  cout<<"Поток запущен"<<endl;
  n += (int)iNum;
  cout<<"Поток завершен"<<endl;
  return 0;
}
```

```
int main()
{
 setlocale(LC_ALL, "Russian");
 int inc =10;
  HANDLE hThread;//Поток
  DWORD IDThread;//Указатель на поток
  cout << "n=" << n << endl;
  // Запускаем поток Add
  hThread = CreateThread(NULL,0,Add,
        (void*)inc,0,&IDThread);
  // Проверяем, что создался
  if(hThread==NULL)
    // Возвращаем ошибку
    return GetLastError();
  // ждем, пока поток Add
  // закончит работу
  WaitForSingleObject(hThread, INFINITE);
  // закрываем дескриптор потока Add
  CloseHandle(hThread);
  cout << "n=" << n << endl;
  return 0;
}
```

Создание потоков

Для создания потоков можно также использовать макрокоманду **`_beginthreadex`**, которая описана в заголовочном файле <process.h> и имеет те же параметры, что и функция **`CreateThread`**.

Как утверждает Джеффри Рихтер в своей книге "Программирование приложений для Windows", использование этой макрокоманды более надежно, чем непосредственный вызов функции CreateThread.

Пример 2. Создание потока макрокомандой _beginthreadex

```
#include <windows.h>
#include <iostream>
#include <string.h>
#include <process.h>
using namespace std;
UINT WINAPI thread(void *pString)
{
int i = 1;
char *pLexema;
pLexema = strtok((char*) pString," ");
while (pLexema != NULL)
{
cout << "Thread find the lexema " << i << " :
"<< pLexema << endl;
pLexema = strtok(NULL, " ") ;
i++;
}
return 0;
}
```

```
int main()
{
char sentence[80];
int i,j,k = 0;
HANDLE hThread;
UINT IDThread;
cout << "Input string: ";
cin.getline(sentence, 80);
j = strlen(sentence);
// создаем поток для подсчета лексем
hThread = (HANDLE)
_beginthreadex(NULL, 0, thread, sentence, 0, &IDThread);
if (hThread == NULL)
return GetLastError();
// сами подсчитываем количество букв "a" в строке
for (i=0; i<j; i++)
if (sentence[i] == 'a')
k++;
cout << "Number of symbols 'a' in the string = " << k << endl;
// ждем окончания разбора на лексемы
WaitForSingleObject(hThread, INFINITE);
// закрываем дескриптор потока thread
CloseHandle(hThread);
return 0;
}
```

Завершение потоков

Функция **ExitThread** – завершить поток:

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

Система посылает DLL процесса сообщение **DLL_THREAD_DETACH**.

Макрокоманда **_endthreadex** (в **process.h**) – завершить поток, запущенный через **_beginthreadex**.

Функция **TerminateThread** – завершить другой поток:

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

В случае успеха возвращает ненулевое значение, иначе — **FALSE**.

Функция **TerminateThread** не освобождает ресурсы, принадлежащие потоку.

Пример 3. Завершение работы потока

```
#include <Windows.h>
#include <iostream>
#include <locale>
using namespace std;

volatile UINT count;

void thread()
{
   for(;;)
   {
      ++count;
      Sleep(100);
   }
}

int main()
{
   setlocale(LC_ALL, "RUSSIAN");
   HANDLE hThread;
   DWORD IDThread;
   char c;
   hThread = CreateThread(NULL, 0,
         (LPTHREAD_START_ROUTINE)thread,
          NULL, 0, &IDThread);
   if(hThread==NULL)
      return GetLastError();
   for(;;)
   {
   cout << "Введите 'y' чтобы показать счет,
или другой символ для завершения потока: ";
      cin >> c;
      if(c=='y')
         cout << "count=" << count << endl;
      else
         break;
   }
   cout << "count=" << count << endl;
   TerminateThread(hThread,0);
   cout << "count=" << count << endl;
   CloseHandle(hThread);
   return 0;
}
```

Приостановка и возобновление потока

Поток имеет счетчик приостановок (СчП), макс.значение = **MAXIMUM_SUSPEND_COUNT**.

Поток исполняется если СчП = 0, иначе поток в подвешенном состоянии.

Функция **SuspendThread** - приостановить исполнение потока:

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

Увеличивает СчП на 1, возвращает текущее значение СчП, или —1.

Чтобы приостановить себя - передать свой псевдодескриптор (можно получить из функции **GetCurrentThread**).

Функция **ResumeThread** - возобновить исполнение потока:

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

если СчП > 0 - уменьшает СчП на 1,

если СчП = 0 - возобновляет поток

если СчП < 0 - оставляет поток в подвешенном состоянии.

Возвращает текущее значение СчП или —1.

Приостановка и возобновление потока

Поток может задержать свое исполнение:

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

sleep = 0 - выполнение потока прерывается, а затем возобновляется при условии, что нет других потоков, ждущих выделения процессорного времени.

sleep = **infinite** - поток приостанавливает свое исполнение навсегда, работа приложения блокируется.

Пример 4. Приостановка и возобновление потока

```
#include <windows.h>
#include <iostream>
using namespace std;
volatile UINT nCount;
volatile DWORD dwCount;
void thread()
{
  for (;;)
  {
     nCount++;
     Sleep(100); //приостанавливаем поток
  }
}
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;
}
```

Псевдодескрипторы потока

Функция **`GetCurrentThread`** - узнать свой дескриптор:

```
HANLDE GetCurrentThread(VOID);
```

Возвращает **псевдодескриптор** текущего потока.

Псевдодескриптор текущего потока отличается от настоящего дескриптора потока:

- может использоваться только самим текущим потоком
- может наследоваться другими процессами
- его не нужно закрывать после его использования
- из него можно получить настоящий дескриптор потока, если его продублировать функцией **`DuplicateHandle`**.

Пример 5. Псевдодескриптор

```
#include <windows.h>
#include <iostream>
using namespace std;

int main ()
{
    HANDLE hThread;
    // получаем псевдодескриптор текущего
потока
    hThread = GetCurrentThread();
    // выводим псевдодескриптор на консоль
    cout << hThread << endl;
    return 0;
}
```

Обработка ошибок в WinAPI

Если функция завершилась неудачей, то код возврата равен **`false`**, **`null`** или — 1.

Внутренний код ошибки (**код последней ошибки** (last-error code)) поддерживается отдельно для каждого потока.

Функция **`GetLastError`** - код последней ошибки:

```
DWORD GetLastError(VOID);
```

Возвращает код последней ошибки, установленной в потоке.

Функция **`SetLastError`** - установить код последней ошибки в потоке:

```
VOID SetLastError(
DWORD dwErrCode // код ошибки
);
```

функция **`FormatMessage`** - получить сообщение, соответствующее коду последней ошибки:

```
DWORD FormatMessage(
DWORD dwFlags, // режимы форматирования
LPCVOID lpSource,  // источник сообщения
DWORD dwMessageId,     // идентификатор сообщения
DWORD dwLanguageId,    // идентификатор языка
LPTSTR lpBuffer,   // буфер для сообщения
DWORD nSize,       // максимальный размер буфера для сообщения
va_list *Arguments     // список значений для вставки в сообщение
);
```

Вопросы для повторения:

1. Последовательность выполнения инструкций во время выполнения программы называется
2. Потоком (thread) называется
3. Объект ядра (последовательность инструкций программы), которому ОС выделяет процессорное время для выполнения приложения, называется
4. Потоку принадлежат следующие ресурсы:
5. Код исполняемой функции, набор регистров процессора, стек для работы приложения, стек для работы операционной системы, маркер доступа, который содержит информацию для системы безопасности, образуют
6. Идентификатор потока, выполняющегося в системе, является
7. В ОС Windows различаются потоки двух типов:
8. Потоки, которые выполняют различные сервисы операционной системы и запускаются ядром операционной системы, являются
9. Потоки, которые служат для решения задач пользователя и запускаются приложением, являются
10. В работающем приложении имеются потоки двух типов:
11. Потоки, которые выполняют различные фоновые задачи в работающем приложении. называются
12. Потоки, которые связаны с окнами и выполняют обработку сообщений, поступающих этим окнам, называются
13. Работающее приложение имеет поток, который является
14. Первичный (primary) (или главный (main)) поток исполняет функцию
15. Поток в работающем приложении создается функцией
16. После завершения потока тот же идентификатор может быть
17. Для создания потоков можно использовать макрокоманду
18. Функция, которая служит для завершения потока:
19. Функция, которая служит для приостановления исполнения потока:
20. Функция, которая служит для возобновления исполнения потока:

Спасибо за внимание!