А.В. Макаров, С.Ю. Скоробогатов, А.М. Чеповский

Учебный курс “CIL и системное программирование в Microsoft .NET”

Лекция 22.

Взаимодействие процессов и потоков

22. Введение

Материалы, рассматриваемые в данной лекции, могут быть разделены на две категории:

- Синхронизация потоков
- Работа с процессами

Большая часть методов синхронизации потоков может быть успешно применена для потоков, принадлежащих разным процессам.

Многие средства взаимодействия потоков, даже основанные на совместном доступе к данным в едином адресном пространстве, могут быть использованы для межпроцессного взаимодействия будучи скомбинированными с общей, разделяемой процессами, памятью.

22.1. Синхронизация потоков
Пример, как не надо делать

```
#include <process.h>
#include <windows.h>
#define THREADS 10
#define ASIZE   10000000
static LONG     array[ASIZE];
unsigned __stdcall ThreadProc( void *param )
{
    for ( int i = 0; i < ASIZE; i++ ) array[i]++;
    return 0;
}
int main( void )
{
   HANDLE  hThread[THREADS]; unsigned  dwThread; int i, errs;
   for ( i = 0; i < THREADS; i++ )
      hThread[i] = (HANDLE)_beginthreadex(
         NULL, 0, ThreadProc, NULL, 0, &dwThread
      );
   WaitForMultipleObjects( THREADS, hThread, TRUE, INFINITE );
   for ( i = 0; i < THREADS; i++ ) CloseHandle( hThread[i] );
   return 0;
}
```

- Прочитать `array[i]`
- Увеличить значение
- Записать `array[i]`

22.1. Синхронизация потоков

Атомарные операции.
Атомарное изменения значения целочисленной переменной.

Критические секции.
Критические секции могут быть использованы только в рамках одного процесса.

Синхронизация с использованием объектов ядра.
Интерфейс синхронизации поддерживают многие объекты ядра, например, файлы, процессы, потоки, консоли, задания и пр., кроме того, Windows предоставляет специальный набор объектов, предназначенный именно для взаимной синхронизации потоков. Объекты ядра могут быть использованы как для синхронизации потоков в рамках одного процесса, так и для синхронизации потоков в разных процессах.

Ожидающие таймеры.
Обеспечивают выполнение операций либо в заданное время, либо с определенной периодичностью. Могут служить либо в качестве объектов синхронизации, чье состояние меняется в указанное время, либо для постановки APC вызова в очередь в нужное время (в этом случае поток долен быть в состоянии ожидания оповещения).

22.1.1. Атомарные операции

ОС Windows предоставляет функции для манипулирования целочисленными 32-х и 64-х разрядными переменными:

Увеличение

`InterlockedIncrement, InterlockedIncrement64`

Уменьшение

`InterlockedDecrement, InterlockedDecrement64`

Изменения значений

`InterlockedExchange, InterlockedExchange64,`
`InterlockedExchangeAdd, InterlockedExchangePointer`

Изменения значений со сравнением

`InterlockedCompareExchange, InterlockedCompareExchangePointer`

```
unsigned __stdcall ThreadProc( void *param )
{
   int    i;
   for (i = 0; i<ASIZE; i++) InterlockedIncrement( array + i );
   return 0;
}
```

22.1.2. Критические секции

Термин «**критическая секция**» обозначает некоторый фрагмент кода, который должен выполняться в исключительном режиме – никакие другие потоки не должны выполнять этот же фрагмент в то же время.

Инициализация структуры `CRITICAL_SECTION`

- `InitializeCriticalSection`
- `InitializeCriticalSectionAndSpinCount`
- `SetCriticalSectionSpinCount`

Освобождение занятых ресурсов

- `DeleteCriticalSection`

Вход в критическую секцию

- `EnterCriticalSection`
- `TryEnterCriticalSection`

Выход из критической секции

- `LeaveCriticalSection`

Критическая секция реализована в пространстве процесса, что делает её самым эффективным средством синхронизации потоков – ожидание может выполняться без переключения в режим ядра.

22.1.2. Критические секции
Пример реализации

```
...
static LONG                array[ASIZE];
static CRITICAL_SECTION    CS;
unsigned __stdcall ThreadProc( void *param )
{
    int    i;
    for ( i = 0; i < ASIZE; i++ ) {
        EnterCriticalSection( &CS );
        array[i]++;
        LeaveCriticalSection( &CS );
    }
    return 0;
}
int main( void )
{
   HANDLE  hThread[THREADS];  unsigned dwThread; int i, errs;
   InitializeCriticalSectionAndSpinCount( &CS, 100 );
   ... /* создание, запуск и ожидание завершения потоков */
   DeleteCriticalSection( &CS );
   return 0;
}
```

Код критической секции будет выполняться потоками монопольно

Число циклов с опросом перед переходом в режим ожидания

22.1.3. Синхронизация с использованием объектов ядра

В Windows для синхронизации используются разные объекты, однако, при рассмотрении синхронизации, особое положение имеет момент перехода ожидаемого объекта в свободное состояние. Поэтому при большом разнообразии объектов, пригодных для синхронизации, существует всего несколько основных функций, осуществляющих ожидание объекта ядра:

```
DWORD  WaitForSingleObject( HANDLE hHandle, DWORD dwMsecs );
DWORD  WaitForMultipleObjects(
  DWORD nCount, HANDLE* lpHandles, BOOL bWaitAll, DWORD dwMsecs
);
```

С точки зрения операционной системы объекты ядра, поддерживающие интерфейс синхронизируемых объектов, могут находиться в одном из двух состояний: **свободном** (*signaled*) и **занятом** (*nonsignaled*). «**Wait**…»-функции ожидают перехода объекта в свободное состояние и, в зависимости от типа объекта, предпринимают необходимые действия.

22.1.3. Синхронизация с использованием объектов ядра

Для возможности обработки APC запросов предусмотрено специальное состояние **ожидания оповещения** (*alertable waiting*).

```
DWORD  WaitForSingleObjectEx(
    HANDLE hHandle, DWORD dwMsecs, BOOL bAlertable
);
DWORD  WaitForMultipleObjectsEx(
    DWORD nCount, HANDLE* lpHandles, BOOL bWaitAll,
    DWORD dwMsecs, BOOL bAlertable
);
DWORD SleepEx( DWORD dwMsecs, BOOL bAlertable );
DWORD SignalObjectAndWait(
    HANDLE hObjectToSignal, HANDLE hObjectToWaitOn,
    DWORD dwMilliseconds, BOOL bAlertable
);
```

Существуют специальные ожидающие функции, расширенные для поддержки GUI интерфейса с обменом сообщениями.

Объекты ядра могут служить средством синхронизации потоков, принадлежащим разным процессам.

22.1.3. Синхронизация с использованием объектов ядра

Объекты ядра, специально разработанные для синхронизации:

События (Event)

Объект, переводимый из занятого состояния в свободное явными вызовами функций. Любой поток может изменять состояние событий.

Семафоры (Semaphore)

Декрементирующие счетчики; свободное состояние – ненулевое значение счетчика.

Мьютексы (Mutex, Mutual **Ex**clusion**)**

«Объекты исключительного владения» - могут принадлежать в данный момент времени только одному потоку; доступ всех остальных потоков к этому объекту блокируется.

22.1.3.1. События

```
HANDLE CreateEvent(
   LPSECURITY_ATTRIBUTES lpEventAttributes,
   BOOL bManualReset, BOOL bInitialState, LPCTSTR lpName
);
BOOL SetEvent( HANDLE hEvent );
BOOL ResetEvent( HANDLE hEvent );
```

```
#define DEFAULT_SECURITY    (LPSECURITY_ATTRIBUTES)NULL
VOID CALLBACK ApcProc( ULONG_PTR dwData ) {
    SetEvent( (HANDLE)dwData );
}
int main( void ) {
  HANDLE    hEvent;
  hEvent = CreateEvent(DEFAULT_SECURITY, TRUE, FALSE, NULL);
  QueueUserAPC(ApcProc, GetCurrentThread(), (ULONG_PTR)hEvent);
  WaitForSingleObject( hEvent, 100 );
  SleepEx( 100, TRUE );
  WaitForSingleObject( hEvent, 100 );
  CloseHandle( hEvent );
  return 0;
}
```

Код завершения WAIT_TIMEOUT

Код завершения WAIT_OBJECT_0

22.1.3.2. Семафоры

```
HANDLE CreateSemaphore(
  LPSECURITY_ATTRIBUTES lpSemaphoreAttributes,
  LONG lInitialCount, LONG lMaximumCount, LPCTSTR lpName
);
BOOL ReleaseSemaphore(
  HANDLE hSemaphore, LONG lReleaseCount, LPLONG lpPreviousCount
);
```

```
#define DEFAULT_SECURITY    (LPSECURITY_ATTRIBUTES)NULL
int main( void ) {
  HANDLE  hSem;
  LONG    lCounter;
  hSem = CreateSemaphore( DEFAULT_SECURITY, 0, 5, NULL );
  ...
  lCounter = 0;
  if ( WaitForSingleObject( hSem, 0 ) == WAIT_OBJECT_0 )
                        ReleaseSemaphore( hSem, 1, &lCounter );
  ...
  CloseHandle( hEvent );
  return 0;
}
```

При успехе счетчик семафора уменьшается

Теперь `lCounter` содержит значение счетчика семафора

22.1.3.3. Мьютексы

```
HANDLE CreateMutex(
    LPSECURITY_ATTRIBUTES lpMutexAttributes,
    BOOL bInitialOwner, LPCTSTR lpName
);
BOOL ReleaseMutex( HANDLE hMutex );
void __cdecl TProc( void *pdata ) {
    WaitForSingleObject( (HANDLE)pdata, 2000 );
    WaitForSingleObject( (HANDLE)pdata, 2000 );
    Sleep( 1000 );
    ReleaseMutex( (HANDLE)pdata );
    ReleaseMutex( (HANDLE)pdata );
}
int main( void ) {
    HANDLE      hMutex, hThread;
    hMutex = CreateMutex( DEFAULT_SECURITY, TRUE, NULL );
    hThread = (HANDLE)_beginthread( TProc, (void*)hMutex );
    Sleep( 1000 );
    ReleaseMutex( hMutex );
    WaitForSingleObject( hThread, INFINITE );
    CloseHandle( hThread );      CloseHandle( hMutex );
    return 0;
}
```

Поток может рекурсивно использовать мьютекс

22.1.4. Ожидающие таймеры

```
HANDLE CreateWaitableTimer(
    LPSECURITY_ATTRIBUTES lpTimerAttributes,
    BOOL bManualReset, LPCTSTR lpTimerName
);

BOOL SetWaitableTimer(
    HANDLE hTimer, const LARGE_INTEGER* pDueTime, LONG lPeriod,
    PTIMERAPCROUTINE pfnCompletion, LPVOID lpArgs, BOOL fResume
);

BOOL CancelWaitableTimer( HANDLE hTimer );
```

```
int main()
{
    HANDLE         hTimer;
    LARGE_INTEGER liDueTime;

    hTimer = CreateWaitableTimer( NULL, TRUE, "WaitTimer" );

    /* задать срабатывание через 5 секунд */
    liDueTime.QuadPart=-50000000;
    SetWaitableTimer( hTimer, &liDueTime, 0, NULL, NULL, 0 );

    WaitForSingleObject( hTimer, INFINITE );
    CloseHandle( hTimer );

    return 0;
}
```

22.1. Выводы

Win32 API предоставляет богатый набор средств для разработки многопоточных приложений. Базовые средства Windows включают эффективную поддержку:

- **Асинхронных операций** ввода-вывода и асинхронных вызовов процедур
- **Потоков уровня ядра** (thread)
- **Потоков уровня процесса** (fiber и использование сообщений)
- **Пулов потоков** и очередей запросов
- Средств для выполнения **атомарных операций**
- Стандартизованных средств внутрипроцессной **синхронизации** (критические секции) и синхронизации с объектами ядра
- **Выполнения операций в заданное время** и с заданным интервалом (ожидающие таймеры)

Большая часть этих средств используется .NET Framework

22.2. Процессы

Межпроцессное взаимодействие

Использование объектов ядра
Рассматривалось при обсуждении взаимодействия потоков и объектов ядра.

Проецирование файлов (file mapping)
Базовых механизмов, связанный с управлением виртуальным адресным пространством процесса и отображением на него файлов.

Использование файловых объектов (pipe, mailslots, sockets)
Базовый механизм, связанный с использованием подсистемы ввода-вывода операционной системы для межпроцессного и межузлового взаимодействия.

Обмен сообщениями (DDE, WM_COPYDATA, …)
Основан на неявном использовании механизма проецирования файлов.

Вызов удаленных процедур (RPC, Remote Procedure Call)
Позволяет описать процедуры, реализованные в других процессах, и обращаться к ним как к обычным процедурам, локальным для данного процесса. Используется как для внутрипроцессного, так и межпроцессного и даже межузлового взаимодействия.

OLE, COM…

22.2.1. Создание процессов

```
BOOL CreateProcess(
    LPCTSTR lpApplicationName, LPTSTR lpCommandLine,
    LPSECURITY_ATTRIBUTES lpProcessAttributes,
    LPSECURITY_ATTRIBUTES lpThreadAttributes,
    BOOL bInheritHandles, DWORD dwCreationFlags,
    LPVOID lpEnvironment, LPCTSTR lpCurrentDirectory,
    LPSTARTUPINFO lpStInfo, LPPROCESS_INFORMATION lpPrInfo
);
//CreateProcessAsUser, CreateProcessWithLogonW, CreateProcessWithTokenW
VOID ExitProcess( UINT uExitCode );
BOOL TerminateProcess( HANDLE hProcess, UINT uExitCode );
```

```
int main( void ) {
    STARTUPINFO  si;   PROCESS_INFORMATION  pi;
    memset( &si, 0, sizeof(si) ); memset( &pi, 0, sizeof(pi) );
    si.cb = sizeof(si);
    CreateProcess(
        NULL, "cmd.exe", DEFAULT_SECURITY, DEFAULT_SECURITY,
        FALSE, NORMAL_PRIORITY_CLASS, NULL, NULL, &si, &pi
    );
    CloseHandle( pi.hThread );
    WaitForSingleObject( pi.hProcess, INFINITE );
    CloseHandle( pi.hProcess );
    return 0;
}
```

22.2.2. Адресное пространство процесса и проецирование файлов

Средства управления адресным пространством

`VirtualAlloc`, **`VirtualFree`**, **`VirtualAllocEx`**, **`VirtualFreeEx`**, **`VirtualLock`** и **`VirtualUnlock`**

Выполняют несколько основных задач:

- Резервирование диапазона адресов.
- Передача этому диапазону физической памяти.

Гранулярность выделения ресурса 64K (**`GetSystemInfo`**)

Проецирование исполняемых файлов

`CreateProcess…`, **`LoadLibrary`**, **`LoadLibraryEx`**

Учитывает внутреннюю структуру файла.

Проецирование файлов данных

Существует возможность проецирования на адресное пространство любого произвольного файла. Для этого необходимо:

- Создать специальный объект ядра, описывающий проекцию файла в память (**`CreateFileMapping`**, **`OpenFileMapping`**)
- Установить соответствие диапазона адресов и фрагмента файла (**`MapViewOfFile`**, **`MapViewOfFileEx`**, **`UnmapViewOfFile`**).

22.2.2. Создание проекции файла

```
HANDLE CreateFileMapping(
    HANDLE hFile, LPSECURITY_ATTRIBUTES lpAttributes,
    DWORD flProtect, DWORD dwMaximumSizeHigh,
    DWORD dwMaximumSizeLow, LPCTSTR lpName
);
LPVOID MapViewOfFile(
    HANDLE hFileMappingObject, DWORD dwDesiredAccess,
    DWORD dwFileOffsetHigh, DWORD dwFileOffsetLow,
    SIZE_T dwNumberOfBytesToMap
);
LPVOID MapViewOfFileEx(
    HANDLE hFileMappingObject, DWORD dwDesiredAccess,
    DWORD dwFileOffsetHigh, DWORD dwFileOffsetLow,
    SIZE_T dwNumberOfBytesToMap, LPVOID lpBaseAddress
);
BOOL UnmapViewOfFile( LPCVOID lpBaseAddress );
```

```
HANDLE hMapping = CreateFileMapping(
    hFile, DEFAULT_SECURITY, PAGE_READWRITE, 0, 256, NULL
);
LPVOID pMapping=MapViewOfFile(hMapping, FILE_MAP_WRITE, 0,0, 0);
...
UnmapViewOfFile( hMapping );
CloseHandle( hMapping );
```

22.2.3. Использование проецирования файлов для межпроцессного взаимодействия

Два процесса могут независимо друг от друга организовать одновременно проецирование одного и того же файла на собственные адресные пространства. Возможны два случая:

- **Процессы открывают один общий файл и создают разные объекты «проекция файла»**. В этом случае Windows не гарантирует когерентности данных двух разных проекций, процессы должны самостоятельно принять меры к согласованию данных и синхронизации доступа.
- **Процессы используют один общий объект «проекция файла»**. В этом случае Windows гарантирует когерентность данных в обеих проекциях. Связывание объекта проекция файла с объектом файл происходит при создании проекции, поэтому процесс, получающий доступ к уже существующей проекции может не иметь никакой информации об используемом файле. Когерентность данных не гарантируется при проецировании файлов с других узлов.

22.2.3. Использование проецирования файлов Пример процесса, создающего проекцию.

```
#include <windows.h>
#define DEFAULT_SECURITY    (LPSECURITY_ATTRIBUTES)NULL
int main( void ) {
   LPVOID                pMapping; int             i,*p;
   PROCESS_INFORMATION pi;         STARTUPINFO   si;
   memset( &si, 0, sizeof(si) ); memset( &pi, 0, sizeof(pi) );
   si.cb = sizeof(si);
   HANDLE hMapping = CreateFileMapping(
      INVALID_HANDLE_VALUE, DEFAULT_SECURITY, PAGE_READWRITE,
      0, 256, "FileMap-AB-874436342"
   );
   pMapping = MapViewOfFile(hMapping, FILE_MAP_WRITE, 0,0, 0);
   CreateProcess(
      NULL, "second.exe", DEFAULT_SECURITY, DEFAULT_SECURITY,
      FALSE, NORMAL_PRIORITY_CLASS, NULL, NULL, &si, &pi
   );
   CloseHandle( pi.hThread );
   for ( p=(int*)pMapping, i=0; i<256; i++ ) *p++=i;
   WaitForSingleObject( pi.hProcess, INFINITE );
   CloseHandle( pi.hProcess );
   UnmapViewOfFile( hMapping );
   CloseHandle( hMapping );
   return 0;
}
```

Заполняем массив начальными значениями.

22.2.3. Использование проецирования файлов Пример процесса, использующего проекцию.

```
#include <windows.h>
int main( void )
{
    LPVOID  pMapping;
    int     i, *p;
    HANDLE hMapping = OpenFileMapping(
        FILE_MAP_READ, FALSE, "FileMap-AB-874436342"
    );
    pMapping = MapViewOfFile( hMapping, FILE_MAP_READ, 0,0, 0 );
    for ( p = (int*)pMapping, i=0; i<256; i++ ) {
        if ( *p++ != i ) break;
    }
    if ( i != 256 ) {
        /* ОШИБКА! */
    }
    UnmapViewOfFile( hMapping );
    CloseHandle( hMapping );
    return 0;
}
```

Проверяем массив.
Строго говоря, здесь нужна синхронизация между процессами, но начальная инициализация второго процесса занимает много больше времени, чем надо первому для заполнения массива.

22.2.4. Межпроцессное взаимодействие с использованием общих секций

Win32 API предусматривает альтернативный декларативный способ межпроцессного взаимодействия с использованием проецирования файлов.

В этом случае специальным образом описанная секция процесса будет отмечена в качестве общей и при проецировании исполняемого файла (образа задачи или разделяемой библиотеки) все экземпляры этого файла(*) будут использовать общую разделяемую проекцию секции.

```
#pragma section("SHRD_DATA",read,write,shared)
```

Задание атрибутов секции

```
__declspec(allocate("SHRD_DATA")) int shared[ 256 ];
```

```
#pragma data_seg("SHRD_DATA")
int shared[ 256 ];
#pragma data_seg()
```

Альтернативные способы задания секции для хранения данных

(*) секции с одинаковыми атрибутами и одинаковыми именами, но описанные в разных исполняемых файлах не будут являться общими.

22.1. Выводы

В основном Win32 API предоставляет два основных механизма межпроцессного взаимодействия:

- На основе проецирования файлов.
- С использованием файловых объектов.

Остальные средства являются более или менее сложными надстройками над этими базовыми механизмами.

Высокоэффективные средства управления адресным пространством и проецированием данных средствами плохо согласуется с управляемой кучей .NET Framework, поэтому явных механизмов для ее использования не предоставляется.

Основные средства межпроцессного взаимодействия .NET базируются либо на файловых объектах, либо на абстракциях более высокого уровня (которые могут быть основаны в том числе на разделяемой памяти).