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

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

Лекция 20.

Общие подходы к реализации приложений с параллельным выполнением операций

Введение

Общие подходы к реализации приложений с параллельным выполнением операций

- Взаимодействие со специализированными устройствами.

  Специфичные для конкретных устройств средства; асинхронный ввод-вывод.

- Средства управления потоками и волокнами.

  Создание, удаление, приостановка и возобновление потоков и волокон; управление характеристиками потоков.

- Взаимодействие потоков в рамках одного процесса.

  Обмен данными через общее адресное пространство; устранение конфликтов при конкурирующем доступе к памяти; управление памятью, локальной для потоков и волокон.

- Взаимодействие между процессами одного компьютера.

  Обмен данными между разными адресными пространствами с использованием общей памяти; дополнительные средства межпроцессного взаимодействия.

- Взаимодействие между процессами разных компьютеров.

  Базовый набор средств для пересылки данных через коммуникационную среду; богатый набор «надстроек» над базовым уровнем.

20.1. Асинхронный ввод-вывод

Асинхронные операции ввода-вывода

С опросом состояния

С ожиданием на объектах ядра

С использованием функций завершения ввода-вывода

```
#include <stdio.h>
int sequential_io( char *filename, char
*buffer, int size )
{
        FILE   *fp;
        int    done;
        fp = fopen( filename, "rb" );
        if ( fp ) {
                done = fread( fp, 1, size, buffer
);
/* выполнение остановлено до завершения чтения
*/
                fclose( fp );
        } else {
                done = 0;
        }
        buffer[done] = '\0';
        return done;
}
```

20.1. Асинхронный ввод-вывод

С опросом состояния

```
OVERLAPPED  ov;  DWORD dwWritten;  BYTE  buffer[ 5000000 ];
HANDLE fh = CreateFile(
    "file.dat", FILE_READ_DATA|FILE_WRITE_DATA,
    FILE_SHARE_READ, (LPSECURITY_ATTRIBUTES)NULL, OPEN_ALWAYS,
    FILE_ATTRIBUTE_NORMAL|FILE_FLAG_OVERLAPPED, NULL
);
ZeroMemory( &ov, sizeof(OVERLAPPED) );  ov.Offset = 12345;
if (
   WriteFile( fh, buffer, sizeof(buffer), &dwWritten, &ov ) ||
   GetLastError() == ERROR_IO_PENDING
) {
   /* код выполняется параллельно с записью данных */
   while (!GetOverlappedResult( fh, &ov, &dwWritten, FALSE) ){}
} else {
   /* возникла ошибка */
}
```

20.1. Асинхронный ввод-вывод

С ожиданием на объектах ядра

```
OVERLAPPED  ov;  DWORD dwWritten;  BYTE  buffer[ 5000000 ];
HANDLE fh = CreateFile(
    "file.dat", FILE_READ_DATA|FILE_WRITE_DATA,
    FILE_SHARE_READ, (LPSECURITY_ATTRIBUTES)NULL, OPEN_ALWAYS,
    FILE_ATTRIBUTE_NORMAL|FILE_FLAG_OVERLAPPED, NULL
);
ZeroMemory( &ov, sizeof(OVERLAPPED) );  ov.Offset = 12345;
ov.hEvent=CreateEvent((LPSECURITY_ATTRIBUTES)NULL,TRUE,FALSE,0);
if (
    WriteFile( fh, buffer, sizeof(buffer), &dwWritten, &ov ) ||
    GetLastError() == ERROR_IO_PENDING
) {
    /* код выполняется параллельно с записью данных */
    GetOverlappedResult( fh, &ov, &dwWritten, TRUE );
} else {
    /* возникла ошибка */
}
```

20.1. Асинхронный ввод-вывод

С использованием функций завершения ввода-вывода

```
OVERLAPPED  ov;  DWORD dwWritten;  BYTE  buffer[ 5000000 ];
HANDLE fh = CreateFile(
    "file.dat", FILE_READ_DATA|FILE_WRITE_DATA,
    FILE_SHARE_READ, (LPSECURITY_ATTRIBUTES)NULL, OPEN_ALWAYS,
    FILE_ATTRIBUTE_NORMAL|FILE_FLAG_OVERLAPPED, NULL
);
ZeroMemory( &ov, sizeof(OVERLAPPED) );  ov.Offset = 12345;
if ( WriteFileEx( fh, buffer, sizeof(buffer), &ov, io_done ) ) {
    /* код выполняется параллельно с записью данных */
    if ( SleepEx( INFINITE, TRUE ) != WAIT_IO_COMPLETION ) {
        /* ввод-вывод пока не завершен, возможно ошибка */
    }
} else {
    /* возникла ошибка */
}
VOID CALLBACK io_done(DWORD dwErr,DWORD dwDone,LPOVERLAPPED lpOv)
{
}
```

20.1. Асинхронный ввод-вывод

Асинхронные операции

С опросом состояния

Обеспечивает самую быструю реакцию на завершение операции, однако ценой повышенной загрузки процессора.

С ожиданием на объектах ядра

Скорость реакция на завершение операции ввода-вывода определяется планировщиком операционной системы, требует перехода в режим ядра для ожидания, однако во время ожидания ресурсов процессора не потребляет.

С использованием функций завершения ввода-вывода

Наиболее трудоёмкий и наименее распространенный в прикладных программах. Является основой для реализации порта завершения ввода-вывода – эффективного средства управления пулом потоков.

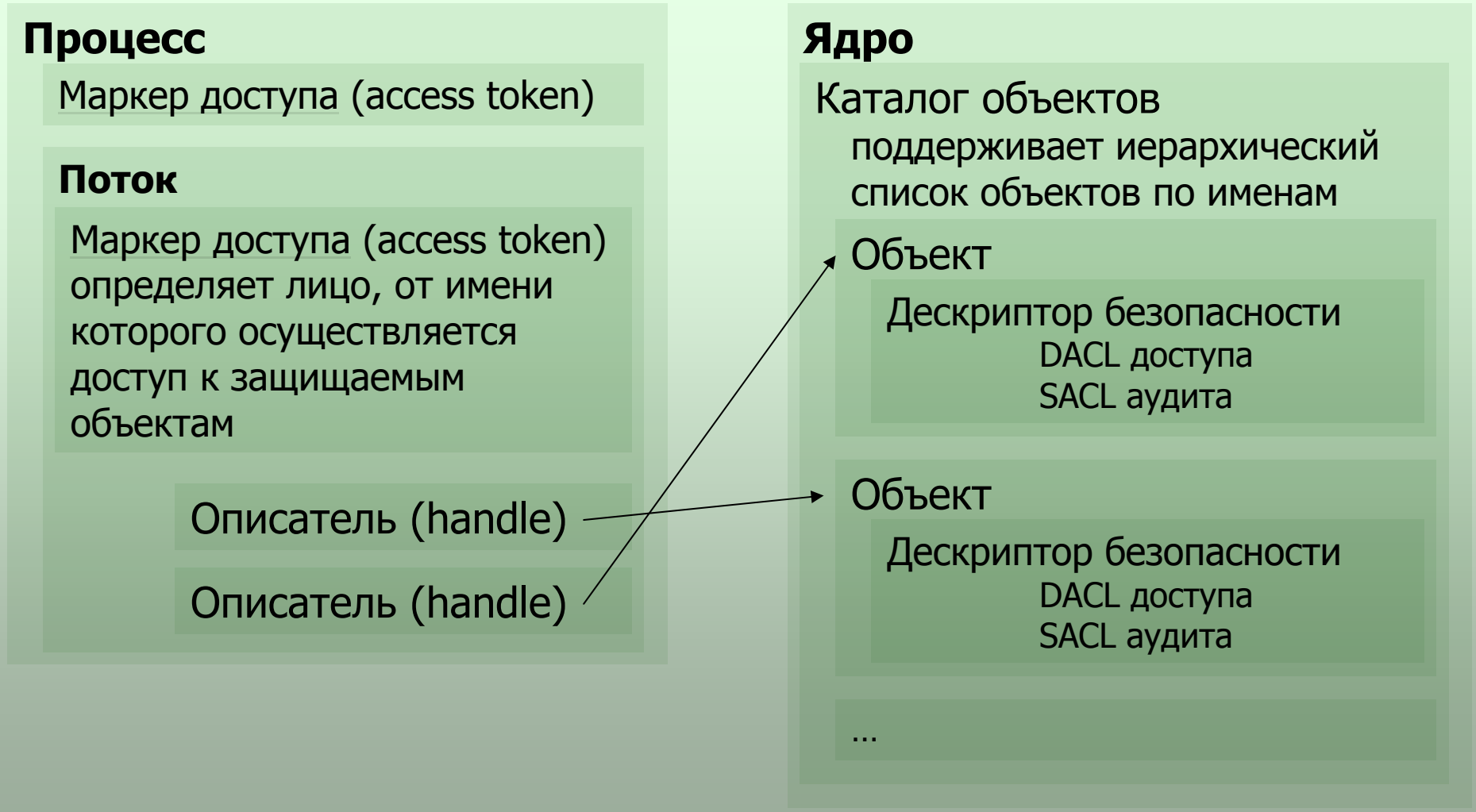
20.2. Асинхронные вызовы процедур

Асинхронные процедуры

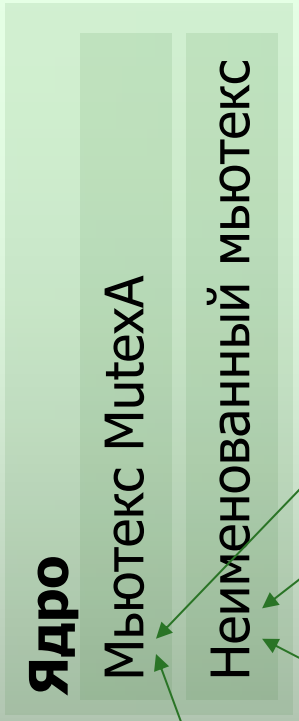
Для реализации асинхронного ввода-вывода система должна поддерживать список адресов процедур, вызываемых в контексте приложения, но асинхронно по отношению к процессу выполнения приложения (APC очередь).
Аналогичный механизм предоставлен разработчикам приложений.

```
#include <windows.h>
VOID CALLBACK ApcProc( ULONG_PTR dwData )
{
    /* … */
}
int main( void )
{
    QueueUserAPC( ApcProc, GetCurrentThread(), 0 );
    /* … */
    SleepEx( 1000, TRUE );
    return 0;
}
```

20.3. Объекты ядра



20.3. Объекты ядра



Процесс 1
идентифицируется описателем hProcess1,
знает описатель второго приложения hProcess2

```
HANDLE hMx1 = CreateMutex( NULL, FALSE, "MutexA" );
```
hMx1 – описатель мьютекса «MutexA» в контексте процесса 1

```
HANDLE hMx2 = CreateMutex( NULL, FALSE, NULL );
```
hMx2 – описатель неименованного мьютекса в контексте процесса 1

```
HANDLE hMx3; /*значение hMx3 в процессе 1 не имеет смысла*/
DuplicateHandle( hProcess1, hMx2, hProcess2, &hMx3,
                 DUPLICATE_SAME_ACCESS, FALSE, 0 );
```

Способ передачи значения не играет роли

Процесс 2
Может использовать значение hMx3 для доступа к неименованному мьютексу

```
HANDLE hMx4 = OpenMutex( MUTEX_ALL_ACCESS, FALSE, "MutexA" );
```
hMx4 – описатель мьютекса «MutexA» в контексте процесса 2

20.3.1. Процессы и потоки

В Windows для идентификации процессов и потоков используют:

- Идентификаторы (DWORD) (выполняют роль уникального имени)
- Описатели объектов ядра (HANDLE)

Процесс:

- Адресное пространство, в котором размещается код и данные
- Контекст безопасности (маркер доступа)
- Дескриптор безопасности (процесс является защищаемым объектом ядра)

Поток:

- Контекст потока
  Содержит сведения о состоянии исполнения потока (стек, регистры и т.д.)
- Дескриптор потока
  Данные, не связанные с состоянием исполнения (приоритет, переменные окружения и пр.)
- Контекст безопасности
  поток может воплощать пользователя, отличного от пользователя процесса
- Дескриптор безопасности

20.3.2. Описатели объектов процесс и поток

получение описателей и идентификаторов

Описатель (HANDLE) ⟶ Идентификатор (DWORD)

```
GetProcessId
GetThreadId
```

Идентификатор (DWORD) ⟶ Описатель (HANDLE)

```
OpenThread
OpenProcess
```

Идентификатор исполняющегося процесса, потока (DWORD)

```
GetCurrentThreadId
GetCurrentProcessId
```

Псевдоописатель исполняющегося процесса, потока (HANDLE)

```
GetCurrentProcess
GetCurrentThread
```

Псевдоописатель (HANDLE) ⟶ Описатель (HANDLE)

```
HANDLE hrealThread;
DuplicateHandle(
    GetCurrentProcess(), GetCurrentProcess(),
    GetCurrentProcess(), &hrealThread,
    DUPLICATE_SAME_ACCESS, FALSE, 0
);
```