ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
НЕФТИ И ГАЗА имени И.М. ГУБКИНА

Кафедра автоматизированных систем управления

Д.Г. ЛЕОНОВ

СИСТЕМНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

Методические рекомендации по
выполнению лабораторных работ

Издательский центр
Москва — 2009

УДК 004.42 (075)

Леонов Д.Г. Системное программное обеспечение. Методические рекомендации по выполнению лабораторных работ. М.: Издательский центр РГУ нефти и газа имени И.М. Губкина, 2009. — 18 с.

Данные методические рекомендации предназначены для использования в рамках курса «Системное программное обеспечение» студентами 3 курса кафедры АСУ. Работы связаны с освоением основных действий, необходимых для организации межпроцессного и межсистемного взаимодействия в ОС семейства Windows. Сборник включает информацию по основным функциям Win32 API, отвечающим за указанные задачи.

Рецензент:
Сарданашвили С.А., д.т.н., профессор кафедры АСУ
РГУ нефти и газа имени И.М. Губкина

Под общей редакцией профессора Григорьева Л.И.

Содержание

Общие требования к лабораторным работам 4
Лабораторная работа №1 4
Задание на работу 4
Справочная информация 5
Лабораторная работа №2 8
Задание на работу 8
Справочная информация 9
Лабораторная работа №3 12
Задание на работу 12
Справочная информация 13
Рекомендуемая литература 17

Общие требования к лабораторным работам

Все лабораторные работы реализовать в VC++ 2005/2008 в виде составного проекта, включающего диалоговое и консольное приложения. Имена классов проектов включают в себя фамилии авторов работ, в настройках проекта отключить использование Unicode.

Диалоговое приложение содержит кнопки "Start", "Stop", "Send", текстовое поле ввода и выпадающий список. Имена классов проектов включают в себя фамилии авторов работ.

Лабораторная работа №1

Задание на работу

При нажатии кнопки "Start" диалоговое приложение запускает консольное приложение. Последующие нажатия кнопки "Start" должны привести к созданию в консольном приложении новых рабочих потоков. Нажатие кнопки "Stop" должно привести к закрытию последнего созданного рабочего потока в консольном приложении, а в случае отсутствия рабочих потоков - к его завершению. Консольное приложение также должно завершиться и при завершении диалогового.

После запуска консоли выпадающий список содержит строки "Все потоки" и "Главный поток", по мере создания новых потоков в него добавляются строки, содержащие их номера (разумеется, при удалении потоков удаляются и соответствующие им строки). Корректировка выпадающего списка осуществляется лишь после получения подтверждения от консоли об успешном создании потоков.

Взаимодействие между приложениями реализовать с помощью объектов ядра *события* (*events*).

Справочная информация

Для запуска внешних приложений в Win32 используется функция CreateProcess:

```
BOOL CreateProcess(
LPCTSTR lpApplicationName,                  // имя программного модуля
LPTSTR lpCommandLine,                       // командная строка
LPSECURITY_ATTRIBUTES lpProcessAttributes,  // атрибуты
LPSECURITY_ATTRIBUTES lpThreadAttributes,   // безопасности
BOOL bInheritHandles,                       // наследование объектов
DWORD dwCreationFlags,                      // флаги создания
LPVOID lpEnvironment,                       // переменные окружения
LPCTSTR lpCurrentDirectory,                 // текущий каталог
LPSTARTUPINFO lpStartupInfo,                // стартовая информация
LPPROCESS_INFORMATION lpProcessInformation // информация о процессе
);
```

Минимально необходимый для работы код:

```
PROCESS_INFORMATION pi;
STARTUPINFO si;
ZeroMemory(&si, sizeof(si))
si.cb = sizeof(si);
CreateProcess(NULL, "prg.exe", NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi);
```

При удачном запуске функция возвращает TRUE, а структура PROCESS_INFORMATION заполняется информацией о запущенном процессе — в частности, поле hProcess содержит описатель процесса.

Для проверки, запущен ли дочерний процесс, можно воспользоваться функцией GetExitCodeProcess, предназначенная для получения кода завершения процесса, который для незавершенных процессов равен значению STILL_ACTIVE.

Помимо этого, завершение процесса совпадает с переходом соответствующего объекта ядра в освободившееся состояние, что можно отследить с помощью функций семейства WaitFor:

```
DWORD WaitForSingleObject(
HANDLE hHandle,                 // описатель ожидаемого объекта ядра
DWORD dwMilliseconds            // время ожидания в миллисекундах
);

DWORD WaitForMultipleObjects(
DWORD nCount,                   // количество ожидаемых объектов ядра
const HANDLE* lpHandles,        // массив описателей
BOOL bWaitAll,                  // признак ожидания освобождения всех объектов
DWORD dwMilliseconds            // время ожидания в миллисекундах
);
```

Первая функция используется для ожидания освобождения одного объекта ядра, вторая — нескольких. В большинстве случаев целесообразно использовать бесконечное время ожидания (INFINITE), за исключением точек ожидания подтверждений. Если параметр bWaitAll равен TRUE, то WaitForMultipleObjects будет ожидать освобождения всех ожидаемых объектов ядра. В противном случае она ожидает освобождения любого объекта из списка, при этом вернет значение, равное сумме WAIT_OBJECT_0 и позиции освобожденного объекта в массиве lpHandles.

Для организации взаимодействия между потоками и процессами в Win32 предусмотрен ряд механизмов, одним из которых является использование объектов ядра ***события***. События создаются с помощью функции CreateEvent:

```
HANDLE CreateEvent(
LPSECURITY_ATTRIBUTES lpSecAttrs,    // атрибуты защиты
BOOL bManualReset,                   // признак ручного сброса
BOOL bInitialState,                  // стартовое состояние
LPTSTR lpName                        // имя объекта
);
```

Уничтожаются события, как и большинство других объектов ядра, с помощью функции CloseHandle.

Переход события в свободное состояние осуществляется с помощью функции SetEvent и семантически соответствует передаче сообщения из точки ее вызова в точку ожидания этого события одной из рассмотренных выше функций семейства WaitFor. При использовании событий с автоматическим сбросом (созданным с параметром bManualReset == FALSE) событие может быть получено только одним ожидающим потоком, при этом оно немедленно переходит в занятое состояние. События с ручным сбросом могут быть одновременно получены произвольным количеством потоков, причем переход в занятое состояние осуществляется вручную с помощью функции ResetEvent.

Для совместного использования одного события в разных процессах можно создать его как именованный объект ядра, передав в качестве последнего параметра одно и то же имя.

Таким образом, ядром консольного приложения становится функция WaitForMultipleObject, заключенная в бесконечный цикл, получающая события, отправляемые диалоговым приложением, обрабатывающая их и отправляющая в ответ события-подтверждения.

Создание новых рабочих потоков в приложениях, использующих MFC, происходит с помощью функции AfxBeginThread:

```
CWinThread* AfxBeginThread(
AFX_THREADPROC pfnThreadProc,                    // потоковая функция
LPVOID pParam,                                   // параметры
int nPriority = THREAD_PRIORITY_NORMAL,          // приоритет потока
UINT nStackSize = 0,                             // размер стека
DWORD dwCreateFlags = 0,                         // флаги создания
потока
LPSECURITY_ATTRIBUTES lpSecAttrs = NULL          // атрибуты защиты
);
```

Потоковая функция объявляется как

```
UINT __cdecl <имя функции>( LPVOID pParam);
```

После выполнения функции AfxBeginThread потоковая функция начинает исполняться параллельно родительской. Для завершения потока могут использоваться функции AfxEndThread, ExitThread, TerminateThread, однако наиболее корректный способ — самостоятельное завершение потоковой функции. Таким образом, для завершения рабочего потока в данной лабораторной работе главный поток консоли должен отправить рабочему сообщение, аналогично тому как диалоговое приложение отправляет сообщения главному потоку консоли. Допустима как реализация, использующая по одному событию с автоматическим сбросом, привязанному к каждому рабочему потоку, так и использующая всего одно событие с ручным сбросом направляемое всем рабочим потокам. В первом случае решение о том, какой поток должен быть закрыт, принимает главная функция консоли, во втором решение принимается самостоятельно каждым потоком.

Лабораторная работа №2

Задание на работу

Добавить в приложения из первой лабораторной работы передачу информации с помощью файла, отображаемого в память (*memory mapped file*). Реализовать транспортные функции в виде динамически подключаемой библиотеки с явной загрузкой.

При нажатии кнопки "Send" диалоговое приложение пересылает консоли текст, введенный в поле ввода. Выпадающий список задает поток-адресат. При этом каждый поток считывает информацию независимо.

Главный поток выводит полученный текст на стандартный вывод (*stdout*), рабочие потоки создают текстовые файлы с именами *<свой номер>.txt* и дописывают в них полученный текст. Файл, отображаемый в

память, защищается от совместного использование с помощью объекта синхронизации *mutex*.

Все действия подтверждаются сообщениями от консоли.

Справочная информация

Отображаемые файлы позволяют использовать механизм виртуальной памяти Win32 для организации ввода/вывода и взаимодействия процессов. После отображения файла в память программа работает с ним как с обычным буфером, в то время как все изменения сохраняются в файле и становятся доступны другим процессам и потокам, имеющим доступ к этому файлу, без дополнительных усилий со стороны программиста.

При работе с отображаемыми файлами можно использовать как произвольный файл на диске, так и системный файл подкачки. В первом случае необходимо получить доступ к файлу с помощью функции CreateFile:

```
HANDLE CreateFile(
LPCTSTR lpFileName,                    // путь до файла
DWORD dwDesiredAccess,                 // желаемый доступ
DWORD dwShareMode,                     // режим совместного использования
LPSECURITY_ATTRIBUTES lpSecAttrs,      // атрибуты защиты
DWORD dwCreationDisposition,           // режим создания
DWORD dwFlagsAndAttributes,            // флаги и атрибуты
HANDLE hTemplateFile                   // описатель шаблона
);
```

Второй параметр является комбинацией значений GENERIC_READ и GENERIC_WRITE, позволяющих дать доступ на чтение и запись. Третий параметр может быть равен нулю (что соответствует запрету совместного использования файла), либо комбинацией значений FILE_SHARE_READ и FILE_SHARE_WRITE. При использовании файла для организации

взаимодействия процессов возможность совместного использования становится существенной. Режим создания определяет поведение функции в ситуациях, когда заданный файл уже создан либо еще не существует, в сочетании с намерением программиста создать новый файл либо получить доступ к существующему.

Полученный описатель файлового объекта передается в качестве параметра в функцию CreateFileMapping, отвечающую за создание отображения файла. Если вместо реально существующего описателя передать значение INVALID_HANDLE_VALUE, будет создана отображение некоторой области системного файла подкачки, что удобно в тех случаях, когда отображение используется исключительно для организации взаимодействия процессов.

```
HANDLE  CreateFileMapping(
HANDLE hFile,                          // описатель файлового объекта
LPSECURITY_ATTRIBUTES lpSecAttrs,      // атрибуты защиты
DWORD flProtect,                       // флаги защиты
DWORD dwMaximumSizeHigh,               // старшее слово максимального
размера
DWORD dwMaximumSizeLow,                // сладшее слово максимального
размера
LPCTSTR lpName                         // имя объекта
);
```

Имя объекта, как и в случае событий, может быть использовано для совместного использования объекта из нескольких процессов. Освобождение и файлового объекта, и объекта отображения происходит с помощью функции CloseHandle. Важно учесть, что при использовании файла подкачки и передачи информации из одного процесса в другой второе отображение с тем же именем должно быть открыто до того как первый процесс закроет соответствующий ему описатель, в противном случае передаваемая информация будет утеряна. Для обеспечения

синхронизации работы необходимо использовать подтверждающие события. Также при использовании файла подкачки необходимо указывать точный размер отображаемых данных.

Непосредственное отображение файла в память осуществляется функцией MapViewOfFile:

```
LPVOID MapViewOfFile(
HANDLE hFileMappingObject,          // описатель объекта отображения
DWORD dwDesiredAccess,              // желаемый доступ
DWORD dwFileOffsetHigh,             // старшее слово смещения
DWORD dwFileOffsetLow,              // младшее слово смещения
SIZE_T dwNumberOfBytesToMap         // размер отображаемой области
);
```

Она возвращает указатель на область памяти, с которой можно работать как с обычным массивом данных. После завершения работы необходимо освободить ресурсы функцией UnmapViewOfFile.

Поскольку при передаче текстовой строки ее размер заранее не известен, а при использовании файла подкачки его необходимо указывать, целесообразно определить структуру фиксированной длины, играющую роль заголовка сообщения и содержащую размер строки вместе с другими параметрами. Например:

```
struct Header
{
        int size;
        int address;
};
```

Далее на принимающей стороне потребуется сначала использовать отображение размером sizeof(Header), что позволит после считывания этой структуры определить размер, необходимый для получения всего сообщения.

Объекты ядра ***mutex*** используются для организации взаимноисключающего доступа к некоторому ресурсу. Для создания мьютекса используется функция CreateMutex:

```
HANDLE CreateMutex(
LPSECURITY_ATTRIBUTES lpSecAttrs,        // атрибуты защиты
BOOL bInitialOwner,                      // признак принадлежности
LPCTSTR lpName                           // имя объекта
);
```

Удаление — традиционное, с помощью CloseHandle.

Ожидание освобождения ресурса с его одновременным занятием производится с помощью вызова функции семейства WaitFor, освобождение — с помощью ReleaseMutex. Освободить мьютекс может только тот поток, который его предварительно занял.

Лабораторная работа №3

Задание на работу

Реализовать функциональность второй лабораторной работы в условиях запуска диалогового и консольного приложений на различных рабочих станциях локальной сети. Сохранить совместимость со второй лабораторной работой с помощью выбора соответствующей динамически подключаемой библиотеки.

Первый вариант (нечетные номера журналу) в качестве транспортного уровня использует механизм именованных каналов (*named pipes*), второй (четные номера) - *Windows Sockets*. Допустима как реализация, использующая прямую доставку сетевых сообщений в консоль, так и использующая промежуточный модуль, принимающий сетевые сообщения и передающий их в консоль, имитируя действия диалогового приложения из предыдущей работы.

Справочная информация

Именованные каналы и сокеты предназначены для организации связи между сервером и одним либо более клиентов. Логика организации взаимодействия в целом сходная.

Именованные каналы создаются на серверной стороне с помощью функции CreateNamedPipe:

```
HANDLE CreateNamedPipe(
LPCTSTR lpName,                 // имя канала
DWORD dwOpenMode,               // режим открытия
DWORD dwPipeMode,               // режим работы канала
DWORD nMaxInstances,            // максимальное количество клиентов
DWORD nOutBufferSize,                       // размеры
DWORD nInBufferSize,                        // буферов
DWORD nDefaultTimeOut,                      // таймаут по умолчанию
LPSECURITY_ATTRIBUTES lpSecAttrs            // атрибуты защиты
);
```

Имя канала имеет вид " \\.\pipe\произвольное имя", в дальнейшем оно используется клиентами для подключения к каналу. Режим открытия определяет, используется ли канал для передачи или получения данных, либо для дуплексной работы. В этот же параметр можно включить дополнительные флаги, определяющие параметры канала, например, асинхронный режим работы. Режим работы канала позволяет выбрать между передачей потока байтов либо отдельных сообщений.

После создания канала сервер переходит в режим ожидания подключений с помощью функции ConnectNamedPipe. При запуске в синхронном режиме эта функция возвращает управление лишь после подключения очередного клиента. Далее описатель канала может использоваться сервером в качестве параметра обычных файловых функций ReadFile и WriteFile в соответствие с используемым протоколом взаимодействия. По завершении работы сервер отключается от канала с помощью функции DisconnectNamedPipe. Закрывается канал, как и прочие

объекты ядра, функцией CloseHandle. Для гарантированной передачи данных из буфера в канал используется функция FlushFileBuffers.

С клиентской стороны подключение к каналу осуществляется с помощью функции CreateFile, которой в качестве имени файла передается имя канала, правила формирования которого описана. Предварительно клиенту целесообразно проверить существование ждущего подключения функцией WaitNamedPipe. Далее, как и не сервере используются функции WriteFile и ReadFile. Для традиционной работы по схеме "отправка запроса"—"получение ответа" на клиентской стороне удобно использовать функции TransactNamedPipe и CallNamedPipe, объединяющие операции ввода/вывода.

Для работы с библиотекой WinSock в MFC используются классы CAsyncSocket и CSocket. Первый из них представляет собой низкоуровневую "обертку", второй позволяет без особых усилий со стороны программиста обеспечить синхронный режим передачи данных. Установка соединения в обоих случаях происходит по одной и той же схеме. Для работы с сокетными классами необходимо подключить заголовочный файл afxsock.h, перед работой библиотека инициализируется функцией AfxSocketInit().

Сначала на стороне сервера создается объект того или иного сокетного класса, после чего вызывается функция Create:

```
BOOL CAsyncSocket::Create(
UINT nSocketPort = 0,
int nSocketType = SOCK_STREAM,
long lEvent=FD_READ|FD_WRITE|FD_OOB|FD_ACCEPT|FD_CONNECT|FD_CLOSE,
LPCTSTR lpszSocketAddress = NULL
);
```

В данном случае можно оставить значения по умолчанию для всех параметров, кроме номера порта, который выбирается более-менее произвольным образом.

Далее серверный сокет переводится в режим ожидания с помощью функции Listen. Само подключение клиентов осуществляется функцией Accept, которой в качестве параметра передается ссылка на непроинициализированный объект сокетного класса.

Дальнейшие прием и передача данных происходят с помощью функций Receive и Send, принимающих в качестве параметров указатель на буфер и его размер:

```
int CAsyncSocket::Receive(const void* lpBuf,  int nBufLen,  int nFlags = 0);
int CAsyncSocket::Send(const void* lpBuf,  int nBufLen,  int nFlags = 0);
```

На стороне клиента после создания объекта сокетного класса подключение к серверу осуществляется функцией Connect, которой передается ip-адрес сервера и порт. В случае успешного соединения клиент, как и сервер, использует функции Send и Receive для передачи и приема данных.

Для еще большего упрощения работы программиста MFC позволяет связать с сокетом объект класса CSocketFile, конструктор которого принимает указатель на CSocket (класс CAsyncSocket для такого взаимодействия непригоден).

Далее объект CSocketFile, являющийся потомком класса CFile, может быть использован либо напрямую, либо через объект класса CArchive, что позволяет использовать при передаче данных через WinSock все средства, предоставляемые механизмом сериализации MFC.

Общая схема взаимодействия клиента и сервера, использующих сериалазацию MFC, выглядит следующим образом:

Сервер

```
CSocket sockSrvr;
sockSrvr.Create(12345);
sockSrvr.Listen();

CSocket sockRecv;
sockSrvr.Accept(sockRecv );
CSocketFile file(&sockRecv);
CArchive arIn(&file, CArchive::load);
CArchive arOut(&file, CArchive::store);

DWORD dwValue;
arIn >> dwValue;
arOut << dwValue;
```

Клиент

```
CSocket sockClient;
sockClient.Create();

sockClient.Connect("127.0.0.1", 12345);


CSocketFile file(&sockClient);
CArchive arIn(&file, CArchive::load);
CArchive arOut(&file, CArchive::store);

DWORD dwValue = 1;
arOut << dwValue;
arIn >> dwValue;
```

Рекомендуемая литература

1. Х.Кастер. Основы Windows NT и NTFS. Microsoft Press Русская редакция, 1996.
2. А.Пол. Объектно-ориентированное программирование на C++. Бином, Невский диалект, 2001.
3. Дж.Рихтер. Windows для профессионалов. Создание эффективных Win32-приложений с учетом специфики 64-разрядной версии Windows. Питер, Русская редакция, 2001.
4. Б.Страуструп. Язык программирования C++. Специальное издания. СПб.;М.: Бином, Невский диалект, 2006.

Леонов Дмитрий Геннадьевич

СИСТЕМНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

Подписано в печать . Формат 60x90/16. Усл. п. л. 1.
Гарнитура Таймс. Бумага офсетная. Печать офсетная
Тираж 100 экз. Заказ №

Издательский центр
РГУ нефти и газа имени И.М. Губкина
119991, Москва, Ленинский проспект, 65
Тел./факс: (499) 233-95-44