Системное программирование

* Получение информации о процессах и потока
* Запуск вторичных потоков. Создание многопоточных приложения
* Синхронизация потоков
* Создание dll
* Работа с реестром Windows

**Процесс**

* Описатель
* Виртуальное адресное пространство 4ГБ
* Потоки исполнения

При запуске процесса система запускает главный поток. Точка начала исполнения главного потока - функция Main(). Главный поток начинает свою работу с этой функции и заканчивает свою работу, выйдя из этой функции. Первичный поток может создавать вторичные потоки. Как только в процессе не остается ни одного работающего потока, такой процесс завершается ОС, т.е. уничтожается.

Для получения информации о процессах используется пространство имен System.Diagnostics.

* Proccess – содержит информацию о процессе
* ProccessModule – содержит информацию о модуле dll, который используется процессом (модули dll загружаются в виртуальное адресное пространство)
* ProcessModuleCollection – коллекция объектов ProccessModule
* ProccessThread – содержит информацию о потоке процесса
* ProccessThreadCollection – коллекция объектов ProccessThread

**Класс Proccess**

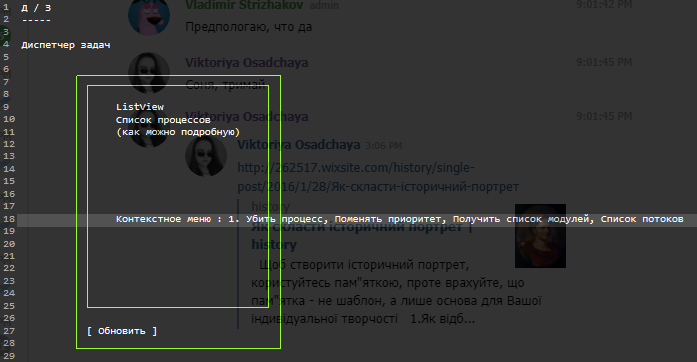
* BasePriority – возвращает базовый приоритет процесса
* ExitTime – время завершения процесса
* Handle – вернет значения описателя
* MachineName – вернет им компьютера, на котором работает процесс
* MainModule – вернет информацию о главном исполняемом модуле процесса
* MainWindowTitle - вернет заголовок главного окна приложения, если есть главное окно
* ProccessModuleCollection – вернет коллекцию с информацией о модулях dll, которые загружены в адресное пространство процесса
* PriorityBoostEnabled - если true, то при получении главным окном фокуса, приоритет процесса будет увеличиваться
* PriorityClass – позволяет задать приоритет, но значения задаются из перечня (Idle, BelowNormal, Normal, AboveNormal, High, RealTime)
* StandardError – стандартный поток для вывода сообщения по ошибке
* StandardInput -
* StandardOutput
* StartTime – возвращает время запуска процесса
* Threads – вернет коллекцию объектов ProccessThread, каждый из них содержит информацию о потоке процесса
* TotalProcessorTime – вернет сколько занимала работа процесса
* UserProcessorTime – вернет количество времени, которое процесс не исполняется в системных библиотеках
* VirtualMemorySize64 – размер виртуальной памяти
* GetProcesses – вернет список процессов, работающих в текущий момент
* GetProcessesByName – вернет процесс по названию
* Kill – завершает процесс
* Start – запустить процесс

**Класс ProcessModule**

* BaseAdress – адрес в виртуальном адресе процесса
* FileName – имя модуля вместе с путем
* ModuleMemorySize – память в байтах, сколько модуль занимает в адресном пространстве
* ModuleName – имя модуля без пути к нему

**Класс ProcessThread**

* StartTime – время запуска потока
* ThreadState – состояние потока (Running, Terminated, Wait)
* WaitReason – причина, по которой поток находится в состоянии ожидания (вернет значение только если ThreadState == Wait)
* CurrentPriority – текущий приоритет



**Создание вторичных потоков.**

Поток – это путь исполнения программы внутри процесса. В процессе может быть множество потоков. При старте процесса ОС запускает первичны поток, который может создавать вторичные потоки. У каждого потока есть идентификатор, хэндл. Также у потока есть область памяти для хранения локальных переменных. Эта область памяти называется стек. Также у потока есть так называемая главная функция потока. Поток начинает свою работу начиная в этой функции и завершает свою работу, выйдя из этой функции.

**Класс Thread**

В C# потоки делятся на фоновые и приоритетные. Разница между ними такая: процесс будет работать до тех пор, пока в нем есть хотя бы один приоритетные поток; как только в процессе не остается ни одного приоритетного потока, даже если в нем остаются фоновые потоки, такой процесс перестает существовать (уничтожается ОС). То есть, наличие фоновых потоков не гарантирует работы процесса. Предполагается, что фоновый поток не играет роли в работе процесса. По умолчанию, все потоки приоритетные.

* Thread(ThreadStart start) – создает объект Thread и принимает delegate на метод вторичного потока
* Thread(ThreadStart start, int maxStackSize) – создает объект Thread, задавая максимальный размер стека
* Thread(ParameterizedThreadStart start) – delegate-метод, который принимает один параметр
* Thread(ParameterizedThreadStart start, int maxStackSize)
* IsAlive – вернет true, если поток исполняется в ОС
* IsBackground – если true, то поток фоновый
* IsThreadPoolThread – вернет true, если поток принадлежит пулу потоков
* Name – позволяет задать или прочитать понятное имя потока
* ThreadPriority – приоритет потока (Lowest, BelowNormal, Normal, AboveNormal, Highest)
* ThreadState – состояние поткоа из перченя (Aborted, Background, Running, Stopped, Unstarted, WaitSleepJoin)
* Abort() – посылает потоку исключительную ситуацию ThreadAbortException
* Interrupt() – прерывает ~~ожидание~~ работу потока, который находится в методах Wait, Sleep, Join
* Join() – приостанавливает исполнение потока, в котором она вызвана, до тех пор, пока поток, для которого она вызвана, не завершит свою работу
* Bool Join(int millisecondsTimeout) – делает то же самое, что и предыдущее, принимая количество миллисекунд, которые главный поток готов ждать до завершения вторичного потока. Возвращает true, если поток завершил свою работу до окончания назначенного времени.
* ~~Resume() – продолжает приостановленный функцией Suspend поток~~
* Static Sleep(int millisecondsTimeout)
* Start() – запускает поток на выполнение
* Start(parameter) – запускает поток на выполнение и принимает параметр типа object
* ~~Suspend() – приостанавливает поток на неопределенное кол-во времени до тех пор, пока не будет вызван метод Resume()~~
* Static bool Yield() – сообщает системе, что поток выполнил задачу, а сейчас ему надо быть приостановленным

**Синхронизация потоков**

Механизмы синхронизации:

1. Монитор 🡪 Критическая секция (не является объектом ядра ОС)
2. Мьютекс 🡪 Мьютекс (является объектом ядра ОС) 🡪 Mutex
3. Семафор 🡪 Семафор (является объектом ядра ОС) 🡪 Semaphore
4. Событие 🡪 Событие (являетя объектом ядра ОС) 🡪 EventWaitHandle

Иерархия классов:

* WaitHandle
* Mutex
* Semaphore
* EventWaitHandle

**Монитор**

Монитор – предназначен для обеспечения монопольного доступа со стороны нескольких потоков к одному общему ресурсу.

* Static Enter(object obj) – вызывается в начале блока кода, в котором необходимо обеспечить монопольный доступ к общему ресурсу.
* Static Exit(object obj) – вызывается в конце блока кода, в котором требуется монопольный доступ к общему ресурсу.

Методы принимают ссылку на объект, который будет идентифицировать эту критическую секцию.



**Mutex**

Mutex – предназначен для монопольного доступ со стороны нескольких потоков к одному общему ресурсу.

Отличия Mutex от Monitor

* У Мьютекса есть владелец – это тот поток, который перевел Mutex в состояние «занят»
* Только владелец Mutex может перевести его в состояние «свободен»
* Если поток-владелец мьютекса занял поток и, не освободив его, завершил свою работу, то такая ситуация называется отказом от мьютекса и ОС в этом случае сама переводит мьютекс в состояние «свободен»

Свойства и методы класса:

* Mutex()
* Mutex (bool initiallyOwned) – позволяет задать состояние Mutex (true – занят)
* Mutex (bool initiallyOwned, string name) – позволяет задать Mutex имя
* Static Mutex OpenExisting (string name) – позволят открыть или создать именованый Mutex
* ReleaseMutes() – переводит мьютекс в состояние «свободен»
* Virtual bool WaiOne() – переводит мьютекс в состояние «занят». Если мьютекс находится в состоянии «свободен», то он переходит в состояние «занят», а поток продолжает свою работу. Если мьютекс уже находится в состоянии «занят», то поток, вызвавший метод WaitOne, приостанавливается до тех пор, пока мьютекс не перейдет в состояние «свободен». В этом случае ожидающие освобождение мьютекса потоки просыпаются, переводит состояние мьютекса в состояние «занят» и продолжает свою работу.
* Virtual bool WaitOne(int millisecondsTimeout) – то же самое, что и предыдущее, только позволяет передать значения тайм-аута в миллисекундах, в течении которого поток готов ждать освобождение мьютекса. Если мьютекс переходит в состояние «свободен» до истечения тайм-аута, то метод возвращает true. Если мьютекс не перешел в состояние свободен, а тайм-аут прошел, то метод возвращает false и поток не становится владельцем мьютекса и продолжает свою работу.

Именованный Мьютекс позволяет синхронизировать потоки в разных процессах.

**Semaphore**

Semaphore – обеспечивает доступ к общему ресурсу со стороны нескольких потоков, ограничивая одновременный доступ количеством потоков.

* Semaphore(int initialCount, int maximumCount) – первый параметр – изначально количество мест, второй параметр – макс. Кол-во мест, которые могут быть в семофоре
* Semaphore(int initialCount, int maximumCount, string name)
* Int Release() – возвращает количество занятых мест перед вызовом этого метода
* Int Release(int releaseCount) – делает то же самое, только позволяет задать кол-во освобождаемых мест
* Virtual bool WaitOne() – занимает одно место в семофоре. Если в семофоре нет свободных мест, то поток, вызвавший этот метод,
* Virtual bool WaitOne(int millisecondTimeout) – то же самое, только задает кол-во времени, в течении которого поток может ждать освобождение в семофоре.

**Событие. Класс EventWaitHandle**

Объект события предназначен для оповещения потоков о наступлении какого-либо события. Объект события может быть как с автоматическим, так и с ручным сбросом. Событие с автоматическим сбросом предназначено для запуска только одного, ожидающего события, потока. Событие с ручным сбросом предназначено для запуска нескольких, ожидающих события, потоков.

* EventWaitHandler(bool initialState, EventResetMode mode) – создает объект события по указанным параметрам: изначальное состояние (true – свободен, false – занят), тип объекта события (AutoReset, ManualReset)
* EventWaitHandler(bool initialState, EventResetMode mode, string name) – то же самое, что и первое, только позволяют задать имя объекта. Именованные объекты предназначены для синхронизации потоков из разных процессов
* Bool Reset() – переводит объект события в состояние «занят». Для режима ручного сброса это единственный способ сделать объект события занятым, т.к. метод WaitOne не переводит объект события в состояние «занят» при ручном сбросе
* Bool Set() – переводит объект события в состояние «свободен», то есть в сигнальное состояние.
* Virtual bool WaitOne() – **для автоматического сброса**: переводит объект события в состояние «занят». Если объект события уже находится в состоянии «занят», то поток вызвавший этот метод, приостанавливается до тех пор, пока объект события не перейдет в состояние «свободен». **Для ручного сброса**: если объект события находится в состоянии «свободен», то метод не приостанавливает работу потока. И при этом не переводит объект события в состояние «занят». Если объект события занят, то метод приостановит работу потока до тех пор, пока объект события не перейдет к состоянию «свободен». Именно за счет того, что метод WaitOne в ручном режиме не делает объект занятым и существует возможность запуска сразу нескольких потоков одновременно.
* Virtual bool WaitOne(int millisecondsTimeout) – то же самое, только ждет заданное время, пока потом освободится

Реестр Windows

Реестр Windows – иерархическая БД которая находится в ОС и предоставляет как приложение, так и самой ОС, сохранять туда всевозможные данные.

* HKEY\_CLASSES\_ROOT – хранит информацию о зарегистрированных типах файлов.
* HKEY\_CURRENT\_USER – хранит настройки текущего пользователя, вошедшего в систему
* HKEY\_LOCAL\_MACHINE – параметры конфигурации, относящиеся к данному компьютеру для всех пользователей
* HKEY\_USERS – содержит информацию о всех пользователях данного компьютера
* HKEY\_CURRENT\_CONFIG – содержит сведения об оборудовании, которое используется локальным компьютером при запуске системы

**Класс Regisrty**

Класс Registy – возвращает нам ссылки на корневые разделы реестра

* Static object GetValue(string keyName, string valueName, object defaultValue)
* Static void SetValue(string keyName, string valueName, object value)

**Класс RegistryKey**

* String Name – название раздела реестра
* Int SubKeyCount – количество подразделов
* Int ValueCount -
* Close() – закрывает раннее открытый раздел реестра
* RegistryKey CreateSubKey(string subkey) – создает подраздел реестра. Если такой объект уже создан, то возвращает ссылку на данный подраздел.
* RegistryKey CreateSubkey(string subkey, bool writeable) – создает или открывает подраздел для записи.
* DeleteSubkey(string subkey) – удаляет раздел
* DeleteSubkey(string subkey, bool throwOnMissingValue) – удаляет раздел, но если такого раздела не существует, то генерирует исключительную ситуацию
* DeleteSubKeyTree(string subKey) – удаляет подраздел и его разделы рекурсивно
* DeleteValue(string name) – удаляет значение
* DeleteValue(string name, book throwOnMissingValue) – удаляет значение, но если такого раздела не существует, то генерирует исключительную ситуацию
* String[] GetSubKeyNames() – возвращает массив с названиями подразделов реестра
* Object GetValue(string name) – удаляет значение
* Object GetValue(String name, object defaultValue) – удаляет значение и вызывает исключение, если значение не существует
* RegistryValueKind GetValueKind(string name) – возвращает тип в данных параметра
* Binary – бинарный
* DWord – int
* ExpandString %root%, %system%
* MultiString – множество строк, разделенных 0, в конце 00
* Qword – int64
* String – строка
* String[] GetValueName() – возвращает массив названий, значений раздела реестра
* RegistryKey OpenSubKey(string name) – открывает существующий раздел реестра
* RegistryKey OpenSubKey(string name, bool writable) – открывает существующий раздел реестра, но указывая для чтения(false) или для записи + чтения(true)
* SetValue(string name, object value) – позволяет установить новую величину для значения в реестре. Если такого значения нет, то оно будет создано.
* SetValue(string name, object value) – то же самое, но указывая тип данных