|  |  |  |
| --- | --- | --- |
| Создание процесса  Параметры CreateProcess  Параметры ***pszApplicationName и pszCommandLine.*** Эти параметры определяют имя исполняемого файла.  Параметры ***psaProcess, psaThread и blnheritHandles .*** Параметры *psaProcess* и *psaThread* определяют нужные атрибуты. В эти параметры можно занести NULL, и система закрепит за данными объектами дескрипторы защиты по умолчанию.  Параметр ***fdwCreate*** определяет флаги, влияющие на создание нового процесса.  Флаг DEBUG\_PROCESS даст возможность родительскому процессу проводить отладку дочернего, а также всех процессов, которые последним могут быть порождены.  Флаг DEBUG\_ONLY\_THIS\_PROCESS аналогичен флагу DEBUG\_PROCESS с тем исключением, что заставляет систему уведомлять родительский процесс о возникновении специфических событий только в одном дочернем процессе — его прямом потомке.  Флаг CREATE\_SUSPENDED позволяет создать процесс и в то же время приостановить его первичный поток Флаг DETACHED\_PROCESS блокирует доступ процессу, инициированному консольной программой, к созданному родительским процессом консольному окну и сообщает системе, что вывод следует перенаправить в новое консольное окно.  Флаг CREATE\_NEW\_CONSOLE приводит к созданию нового консольного окна для нового процесса. Имейте в виду, что одновременная установка флагов CREATE\_NEW\_CONSOLE и DETACHED\_PROCESS недопустима.  Флаг CREATE\_NO\_WINDOW не дает создавать никаких консольных окон для данного приложения и тем самым позволяет исполнять его без пользовательского интерфейса.  Флаг CREATE\_BREAKAWAY\_FROM\_JOB позволяет процессу, включенному в задание, создать новый процесс, отделенный от этого задания  Параметр *fdwCreate* разрешает задать класс приоритета процесса:  Idle (простаивающий) IDLE\_PRIORITY\_CLASS  Below normal (ниже обычного) BELOW\_NORMAL\_PRIORITY\_CLASS  Normal (обычный) NORMAL\_PRIORITY\_CLASS  Above normal (выше обычного) ABOVE\_NORMAL\_PRIORITY\_CLASS  High (высокий) HIGH\_PRIORITY\_CLASS  Realtime (реального времени) REALTIME\_PRIORITY\_CLASS  Классы приоритета влияют на распределение процессорного времени между процессами и их потоками.  Параметр ***pvEnvironment*** указывает на блок памяти, хранящий строки переменных окружения. Обычно вместо этого параметра передается NULL.  Параметр ***pszCurDir*** позволяет родительскому процессу установить текущие диск и каталог для дочернего процесса. Если его значение — NULL, рабочий каталог нового процесса будет тем же, что и у приложения, его породившего.  Параметр ***psiStartlnfo*** указывает на структуру STARTUPINFO. Элементы структуры STARTUPINFO используются Windows-функциями при создании нового процесса.  Параметр ***ppiProclnfo*** указывает на структуру PROCESS\_INFORMATION, которую Вы должны предварительно создать; ее элементы инициализируются самой функцией CreateProcess.  Завершение процесса  Существует 4 гипотетических варианта завершения процесса.  входная функция первичного потока возвращает управление (рекомендуемый способ);  один из потоков процесса вызывает функцию ExitProcess (нежелательный способ);  поток другого процесса вызывает функцию TerminateProcess (тоже нежелательно);  все потоки процесса умирают (большая редкость).  Явный вызов ExitProcess и TerminateProcess – распространенная ошибка, которая мешает правильной очистке ресурсов.  Создание потока  HANDLE CreateThread  ***Параметр psa*** является указателем на структуру SECURITY\_ATTRIBUTES Если Вы хотите, чтобы объекту ядра "поток" были присвоены атрибуты защиты по умолчанию (что чаще всего и бывает), передайте в этом параметре NULL A чтобы дочерние процессы смогли наследовать описатель этого объекта, определите структуру SECURI TY\_ATTRIBUTES и инициализируйте ее элемент *hlnheritHandle* значением TRUE.  ***Параметр cbStack*** определяет, какую часть адресного пространства поток сможет использовать под свой стек. Каждому потоку выделяется отдельный стек.  **Параметр *pfnStartAddr*** определяет адрес функции потока, с которой должен будет начать работу создаваемый поток, а **параметр *pvParam*** идентичен параметру *рvРаrаm* функции потока. *CreateTbread* лишьпередает этот параметр по эстафете той функции, с которой начинается выполнение создаваемого потока. Таким образом, данный параметр позволяет передавать функции потока какое-либо инициализирующее значение. Оно может быть или просто числовым значением, или указателем на структуру данных с дополнительной информацией. Вполне допустимо и даже полезно создавать несколько потоков, у которых в качестве входной точки используется адрес одной и той же функции. Например, можно реализовать Web-сервер, который обрабатывает каждый клиентский запрос в от дельном потоке. При создании каждому потоку передается свое значение *рvParam.*  ***Параметр fdwCreate*** определяет дополнительные флаги, управляющие созданием потока. Он принимает одно из двух значений. 0 (исполнение потока начинается немедленно) или CREATE\_SUSPENDED. В последнем случае система создает поток, инициализирует его и приостанавливает до последующих указаний. Флаг CREATE\_SUSPENDED позволяет программе изменить какие-либо свойства потока перед тем, как он начнет выполнять код.  ***Параметр pdwThreadlD****—* это адрес переменной типа DWORD, в которой функция возвращает идентификатор, приписанный системой новому потоку. В Windows 2000 и Windows NT 4 в этом параметре можно передавать NULL (обычно так и делается). Тем самым Вы сообщаете функции, что Вас не интересует идентификатор потока в Windows 95/98 это приведет к ошибке, так как функция попытается записать идентификатор потока по нулевому адресу, что недопустимо. И поток не будет создан.  Функция CreateRemoteThread создает поток, который запускается в виртуальном адресном пространстве другого процесса.  Завершение потока  Поток можно завершить четырьмя способами:  функция потока возвращает управление (рекомендуемый способ);  поток самоуничтожается вызовом функции ExitThread (нежелательный способ);  один из потоков данного или стороннего процесса вызывает функцию TerminateThread (нежелательный способ);  завершается процесс, содержащий данный поток (тоже нежелательно).  Явный вызов *ExitThread* и *TerminateThread* нежелателен, т.к.процесс продолжает работать, но при этом весьма вероятна утечка памяти или других ресурсов.  *Возврат управления функцией потока*  При этом:  любые С++-объекты, созданные данным потоком, уничтожаются соответствующими деструкторами;  система корректно освобождает память, которую занимал стек потока;  система устанавливает код завершения данного потока (поддерживаемый объектом ядра "поток”) –его и возвращает Ваша функция потока;  счетчик пользователей данного объекта ядра "поток" уменьшается на 1.  *Функция ExitThread*  Поток можно завершить принудительно, вызвав:  *VOID ExitThread(DWORD dwExitCоde);*  В параметр *dwExitCode* Вы помещаете значение, которое система рассматривает как код завершения потока.  *Функция TerminateThread*  Вызов этой функции также завершает поток:  BOOL TerminateThread( HANDLE hThread, DWORD dwExitCode);  В параметр *dwExitCode* помещается код завершения потока. После того как поток будет уничтожен, счетчик пользователей его объекта ядра "поток” уменьшится.  *Если завершается процесс*  Функции *ExitProcess* и *TerminateProcess* принудительно завершают потоки, принадлежащие завершаемому процессу.  Эти функции прекращают выполнение всех потоков, принадлежавших завершенному процессу. При этом гарантируется высвобождение любых выделенных процессу ресурсов, в том числе стеков потоков. Однако эти две функции уничтожают потоки принудительно так, будто для каждого из них вызывается функция *TerminateThread.* А это означает, что очистка проводится некорректно, деструкторы С++-объектов не вызываются, данные на диск не сбрасываются и т. д.  *Действия при завершении потока*  Освобождаются все описатели User-объектов, принадлежавших потоку.  Код завершения потока меняется со STILL\_ACTIVE на код, переданный в функцию *ExitThread* или *TerminateThread.*  Объект ядра "поток" переводится в свободное состояние.  Если данный поток является последним активным потоком в процессе, завершается и сам процесс.  Счетчик пользователей объекта ядра "поток" уменьшается на 1.  *BOOL GetExitCodeThread( HANDLE hThread, PDWORD pdwExitCode);*  Управление динамическими приоритетами потока  BOOL SetThreadPriorityBoost(  HANDLE hThread,           // дескриптор потока  BOOL DisablePriorityBoost // состояние //форсирования приоритета  );  BOOL GetThreadPriorityBoost(  HANDLE hThread,             // дескриптор потока  PBOOL pDisablePriorityBoost // состояние форсажа //приоритета  );  Поток должен иметь право доступа **THREAD\_SET\_INFORMATION**.  Управление потоками  Флаг CREATE\_SUSPENDED  Если поток создан с флагом CREATE\_SUSPENDED, то после своего создания он остается в приостановленном состоянии. Вы можете настроить некоторые его свойства (например, приоритет, о котором мы поговорим позже). Закончив настройку, Вы должны разрешить выполнение потока. Для этого вызовите *ResumeThread* и пере дайте описатель потока, возвращенный функцией *CreateThread*.  **DWORD ResumeThread(HANDLE hThread);**  Выполнение потока можно приостановить не только при его создании с флагом CREATE\_SUSPENDED, но и вызовом *SuspendThread.* Выполнение отдельного потока можно приостанавливать несколько раз. Если поток приостановлен 3 раза, то и возобновлен он должен быть тоже 3 раза — лишь тогда система выделит ему процессорное время.  **DWORD SuspendThread(HANDLE hThread);**  Засыпание и переключение потоков  VOID Sleep (  DWORD dwMilliseconds  );  Эта функция приостанавливает поток па *dwMilliseconds* миллисекунд. Отметим несколько важных моментов, связанных с функцией *Sleep.*  Вызывая *Sleep,* поток добровольно отказывается от остатка выделенного ему кванта времени  Система прекращает выделять потоку процессорное время на период, *пример но* равный заданному, Все верно: если Вы укажете остановить поток на 100 мс, приблизительно на столько он и "заснет", хотя не исключено, что его сон про длится на несколько секунд или даже минут больше. Вспомните, Windows не является системой реального времени. Ваш поток может возобновиться в заданный момент, но это зависит от того, какая ситуация сложится в системе к тому времени.  Вы можете вызвать *Sleep* и передать в *dwMilliseconds* значение INFINITE, вообще запретив планировать поток. Но это не очень практично — куда лучше корректно завершить поток, освободив его стек и объект ядра.  Вы можете вызвать *Sleep* и передать в *dwMilliseconds* нулевое значение. Тогда Вы откажетесь от остатка своего кванта времени и заставите систему подключить к процессору другой поток. Однако система может снова запустить Ваш поток, если других планируемых потоков с тем же приоритетом нет.  BOOL SwitchToThread();  Функция *SwitchToThread* позволяет подключить к процессору другой поток (если он есть).  Вызов *SwitchToThread* аналогичен вызову *Sleep с* передачей в *dwMilliseconds* нулевого значения. Разница лишь в том, что *SwitchToThread* даетвозможность выполнять потоки с более низким приоритетом, которым не хватает процессорного времени, а *Sleep* действует без оглядки на "голодающие" потоки.  Определение периодов выполнения потока  BOOL GetThreadTimes(  HANDLE hThread,  PFILETIME pftCreationTime,  PFILETIME pftExitTime,  PFILETIME pftKernelTime,  PFILETIME pftUserTime );  С помощью этой функции можно определить время, необходимое для выполнения сложного алгоритма.  *GetThreadTimes* не годится для высокоточного измерения временных интервалов.  *GetThreadTimes* возвращает четыре временных параметра:  Время coздания (creation time) – Абсолютная величина, выраженная в интервалах по 100 нс. Отсчитывается с полуночи 1 января 1601 года по Гринвичу до момента создания потока.  Время завершении (exit time) – Абсолютная величина, выраженная в интервалах по 100 нс Отсчитывается с полуночи 1 января 1601 года по Гринвичу до момента завершения потока. Если поток все еще выполняется, этот показатель имеет неопределенное значение.  Время выполнения ядра (kernel time) – Относительная величина, выраженная в интервалах по 100 нс. Сообщает время, затраченное этим потоком на выполнение кода операционной системы.  Время выполнения User (User time) – Относительная величина, выраженная в интервалах по 100 нc. Сообщает время, затраченное потоком на выполнение кода приложения.  С помощью этой функции можно определить время, необходимое для выполнения сложного алгоритма.  *GetThreadTimes* не годится для высокоточного измерения временных интервалов. |  |  |