ОГЛАВЛЕНИЕ

[Часть 1 Windows (Процессы и потоки) 3](#_Toc95130886)

[Что такое процесс? 3](#_Toc95130887)

[Адресное пространство 3](#_Toc95130888)

[Системные ресурсы 4](#_Toc95130889)

[Объект-процесс 5](#_Toc95130890)

[Что такое поток? 8](#_Toc95130891)

[Многозадачность и многопроцессорная обработка 8](#_Toc95130892)

[Многопоточность 11](#_Toc95130893)

[Объект-поток 13](#_Toc95130894)

[Планирование потоков 15](#_Toc95130895)

[Квант 15](#_Toc95130896)

[Сценарии планирования 16](#_Toc95130897)

[Самостоятельное переключение 16](#_Toc95130898)

[Вытеснение 16](#_Toc95130899)

[Завершение кванта 16](#_Toc95130900)

[Завершение потока 16](#_Toc95130901)

[Динамическое повышение приоритета 16](#_Toc95130902)

[Абстрагирование приоритетов 17](#_Toc95130903)

[Лабораторная работа 2 Процессы и потоки (программирование) 17](#_Toc95130904)

[Теория 17](#_Toc95130905)

[Создание процесса 18](#_Toc95130906)

[Параметры lpApplicationName и lpCommandLine 18](#_Toc95130907)

[Параметры lpProcessAttributes, lpThreadAttributes и bInheritHandeles 19](#_Toc95130908)

[Параметр DwCreationFlags 19](#_Toc95130909)

[Параметр lpEnvironment 19](#_Toc95130910)

[Параметр lpCurrentDirectory 19](#_Toc95130911)

[Параметр lpStartupInfo 20](#_Toc95130912)

[Параметр lpProcessInformation 20](#_Toc95130913)

[Создание потока 21](#_Toc95130914)

[Параметр lpThreadAttributes 22](#_Toc95130915)

[Параметр dwStackSize 22](#_Toc95130916)

[Параметры lpStartAddress и lpParameter 22](#_Toc95130917)

[Параметр dwCreationFlags 22](#_Toc95130918)

[Параметр lpThreadId 22](#_Toc95130919)

[Чему нужно научиться 22](#_Toc95130920)

[Задания 23](#_Toc95130921)

[Уровень 1 (A) 23](#_Toc95130922)

[Уровень 2 (A) 23](#_Toc95130923)

[Уровень 3 (A) 23](#_Toc95130924)

[Уровень 1 (В) 23](#_Toc95130925)

[Уровень 2 (В) 23](#_Toc95130926)

[Уровень 3 (В) 24](#_Toc95130927)

СПИСОК РИСУНКОВ и ТАБЛИЦ

[Рис. 1 Виртуальная и физическая память 4](#_Toc95130928)

[Рис. 2 Процесс и его ресурсы 5](#_Toc95130929)

[Рис. 3 Блоки переменных окружения процесса (PEB) и потока (TEB) 6](#_Toc95130930)

[Рис. 4 Блоки процесса исполнительной системы (EPROCESS) и ядра (KPROCESS) 7](#_Toc95130931)

[Рис. 5 Состояния потоков 10](#_Toc95130932)

[Рис. 6 Многозадачность 10](#_Toc95130933)

[Рис. 7 Компилятор, состоящий из двух процессов 11](#_Toc95130934)

[Рис. 8 Мультипроцессорная обработка 11](#_Toc95130935)

[Рис. 9 Многопоточный компилятор 12](#_Toc95130936)

[Рис. 10 Многопоточный сервер 13](#_Toc95130937)

[Рис. 11 Объект-поток 14](#_Toc95130938)

[Рис. 12 Соотношение приоритетов 15](#_Toc95130939)

[Таблица 1 Относительные приоритеты потоков 17](#_Toc95130940)

# Часть 1 Windows (Процессы и потоки)

В однозадачной ОС пользователь может запустить одновременно не более одной программы и только после того как она закончит работу можно будет запустить другую. В многозадачной ОС пользователи могут одновременно запускать несколько программ или даже несколько копий одной программы.

В чем разница между программой и процессом? Программа – это статическая последовательность команд, а ***процесс*** (***process***) – это программа и системные ресурсы, необходимые для ее выполнения. Процесс является субъектом владения ресурсами и единицей работы. ОС выделяет каждому процессу порцию системных ресурсов и гарантирует, что программа каждого процесса будет направляться на исполнение в определенном порядке и своевременно.

ОС содержит блок кода, управляющий созданием и удалением процессов, а также отношениями между ними. Этот код называется ***структурой процессов*** (***process structure***) и в Windows NT реализован ***диспетчером процессов*** *(****process manager****)*.

Его основная задача предоставить набор базовых сервисов процесса, которые подсистемы среды могли бы использовать для эмуляции своих собственных уникальных структур процессов.

В разных ОС процессы реализованы по-разному. Они различаются своим представлением (структурами данных), способами именования и защиты, а также отношениями между собой. Базовые процессы Windows NT имеют ряд характеристик, отличающих их от процессов других ОС:

* процессы реализованы как объекты, и доступ к ним осуществляется посредством объектных сервисов;
* в адресном пространстве процесса может исполняться несколько потоков;
* объект-процесс и объект-поток имеют встроенные возможности синхронизации.

## Что такое процесс?

Процесс состоит из:

* исполняемой программы (код и данные);
* закрытого ***адресного пространства*** *(****address space****)*, т.е. набора адресов виртуальной памяти, который процесс может использовать;
* ***системных ресурсов***, выделяемых ОС процессу во время выполнения программы (семафоров, файлов и т.д.);
* по крайней мере, одного ***потока управления*** *(****thread of execution****).* Поток – это сущность внутри процесса, которую ядро NT направляет на исполнение. Без него программа процесса не может выполняться.

### Адресное пространство

С помощью *системы виртуальной памяти (virtual memory)* программисты (и создаваемые ими процессы) получают логический образ памяти, который не совпадает с ее физической структурой (см. Рис. 1 Виртуальная и физическая память).

При всяком обращении процесса по виртуальному адресу система виртуальной памяти транслирует этот адрес в физический адрес. Она также предотвращает непосредственный доступ процесса к виртуальной памяти, занятой другими процессами или ОС. Для исполнения кода ОС или доступа к памяти ОС поток должен выполняться в *режиме ядра (kernel mode)*. Большинство процессов – это процессы *пользовательского режима* (*user mode*).

Поток пользовательского режима получает доступ к ОС, вызывая некоторый системный сервис. Когда поток вызывает сервис, происходит переключение из пользовательского режима в режим ядра. ОС проверяет аргументы, переданные сервису, после чего исполняет сервис. Перед возвратом управления пользовательской программе ОС переключает поток обратно в пользовательский режим.



Рис. 1 Виртуальная и физическая память

### Системные ресурсы

Кроме закрытого адресного пространства, с каждым процессом связан набор системных ресурсов. Это показано на Рис. 2 Процесс и его ресурсы.



Рис. 2 Процесс и его ресурсы

***Маркер доступа*** (***Access Token***) присоединяет к процессу ОС. Это объект исполнительной системы, который содержит информацию о правах зарегистрированного в системе пользователя, которого представляет данный процесс. Если процессу требуется получить информацию о своем маркере доступа или изменить некоторые атрибуты маркера, он должен открыть описатель своего объекта-маркера. Подсистема защиты определяет, есть ли у объекта такое право.

Ниже маркера доступа расположен набор структур данных, созданный диспетчером виртуальной памяти для отслеживания виртуальных адресов, используемых процессом.

Таблица объектов показана внизу. Процесс открыл описатели потока, файла и секции совместно используемой памяти. Описание виртуального адресного пространства содержит информацию о виртуальных адресах, занятых стеком потока и объектом-секцией.

### Объект-процесс

Каждый процесс в Windows NT представлен ***блоком процесса***, создаваемым исполнительной системой (***EPROCESS***). В блоке EPROCESS содержатся атрибуты процесса и указатели на некоторые структуры данных. Так, у каждого процесса есть один или более потоков, представляемых ***блоками потоков*** исполнительной системы (***ETHREAD***). Блок EPROCESS и связанные с ним структуры данных хранятся в системном пространстве. Исключение составляет только ***блок переменных окружения процесса*** (***process environment block, PEB***), он находится в адресном пространстве процесса (см. Рис. 3 Блоки переменных окружения процесса (PEB) и потока (TEB)).



Рис. 3 Блоки переменных окружения процесса (PEB) и потока (TEB)

В исполнительной системе процессы – это объекты исполнительной системы, создаваемые и уничтожаемые диспетчером объектов. Объект-процесс, как и другие объекты, содержит заголовок, создаваемый и инициализируемый диспетчером объектов. В заголовке хранятся стандартные атрибуты объекта, такие как дескриптор защиты объекта, имя и каталог объектов, в котором хранится имя, если оно есть.

Диспетчер процессов определяет атрибуты, хранящиеся в теле объектов-процессов, а также предоставляет системные сервисы для чтения и изменения этих атрибутов. Атрибуты и сервисы для объектов-процессов показаны на Рис. 4 Блоки процесса исполнительной системы (EPROCESS) и ядра (KPROCESS). Объект процесс исполнительной системы включает объект процесс ядра (содержит указатель на объект процесс ядра). Ядро управляет объектом процесс ядра, а исполнительная система управляет объектом исполнительной системы.

Мы рассматриваем основные атрибуты этих объектов по отдельности, для того чтобы лучше понять какие задачи решаются ядром, а какие исполнительной системой. При выполнении лабораторных работ эта детализация не существенна и надо будет использовать Win32 API функции для работы с объектом процесс исполнительной системы.



Рис. 4 Блоки процесса исполнительной системы (EPROCESS) и ядра (KPROCESS)

Рассмотрим основные атрибуты:

* *идентификатор процесса* – уникальное значение, идентифицирующее процесс в ОС;
* *базовый приоритет* - базовый приоритет потоков процесса;
* *привязка к процессорам (процессорное сродство)* – набор процессоров, на которых потоки процесса могут исполняться по умолчанию;
* *размеры квот* – максимальный объем резидентной и нерезидентной системной памяти, пространства в файле подкачки и процессорного времени, выделяемый пользовательскому процессу;
* *статус завершения* – причина завершения процесса.

Некоторые атрибуты объекта-процесса налагают ограничения на потоки, исполняемые внутри процесса. Например, на многопроцессорном компьютере процессорное сродство может ограничить исполнение потоков процесса только на заданных процессорах. Аналогично, размеры квот регулируют, сколько памяти, пространства в файле подкачки и процессорного времени могут использовать все потоки процесса вместе.

Базовый приоритет процесса помогает ядру NT регулировать приоритет потоков в системе. Приоритеты потоков изменяются, но всегда остаются в диапазоне базовых приоритетов их процессов.

Большинство сервисов объекта-процесса мы будем использовать при выполнении лабораторных работ. Например, сервис завершения процесса останавливает исполнение всех его потоков, закрывает все открытые описатели объектов и уничтожает виртуальное адресное пространство процесса.

## Что такое поток?

Поток – это единица исполнения, отдельный счетчик команд или подлежащая планированию сущность внутри процесса.

В то время как процесс – это логическое представление работы, которую должна выполнить ОС, поток отображает одну из, возможно, многих необходимых подзадач. Предположим, что пользователь запустил приложение для работы с базой данных. ОС представляет этот вызов приложения как один процесс. Пусть теперь пользователь запросил генерацию отчета по данным из базы и сохранение этого отчета в файле. Пока идет выполнение этой длительной операции, пользователь ввел новый запрос к базе данных. ОС представляет каждый из запросов – генерацию отчета и новый запрос к базе – как отдельные потоки внутри процесса приложения для работы с базой данных. Эти потоки могут выполняться процессором независимо друг от друга, т.е. обе операции можно выполнять в одно и то же время (параллельно).

Основные составляющие потока в исполнительной системе NT:

* Уникальный идентификатор, называемый идентификатором клиента
* Содержимое набора регистров, отражающее состояние процессора
* Два стека: один используется потоком при работе в пользовательском режиме, а другой - в режиме ядра
* Собственная область памяти, предназначенная для использования подсистемами, библиотеками периода выполнения и динамически подключаемыми библиотеками (DLL).

Регистры, стек, и собственная область памяти называются *контекстом (context)* потока. Фактически данные, составляющие контекст потока, определяются типом процессора.

Поток находится в адресном пространстве процесса, используя его для хранения данных во время выполнения. Если в одном процессе существует несколько потоков, то они совместно используют адресное пространство и все ресурсы, включая маркер доступа, базовый приоритет и описатели объектов из таблицы объектов процесса. Ядро NT направляет потоки на исполнение некоторому процессору. Таким образом, каждый процесс NT должен иметь, по меньшей мере, один поток.

### Многозадачность и многопроцессорная обработка

ОС вытесняющей многозадачностью должна использовать тот или иной алгоритм, позволяющий ей распределять процессорное время между потоками. Каждые 20 мс Windows просматривает все существующие объекты потоки и отмечает те из них, которые могут получить процессорное время. Далее она выбирает один из таких объектов и загружает в регистры процессора значение его контекста. Эта операция называется переключением контекста (context switching). Поток выполняет код и манипулирует данными в адресном пространстве своего процесса. Примерно через 20 мс Windows сохранит значения регистров процессора в контексте потока и приостановит его выполнение. Далее система просмотрит остальные объекты потоки, подлежащие выполнению, выберет один из них, загрузит его контекст в регистры процессора, и все повторится. Этот цикл операций – выбор потока, загрузка его контекста, выполнение и сохранение контекста – начинается с момента запуска системы и продолжается до ее выключения (см. Рис. 5 Состояния потоков).

Система планирует выполнение только тех потоков, которые могут получить процессорное время. У некоторых объектов-потоков значение счетчика простоев (suspend count) больше 0, это значит, что соответствующие потоки приостановлены и не получают процессорного времени. Кроме приостановленных, существуют и другие потоки, не участвующие в распределении процессорного времени, - они ожидают каких-либо событий.

Поток получает доступ к процессору на 20 мс, после чего планировщик переключает процессор на выполнение другого потока. Но так происходит, только если у всех потоков один приоритет. На самом деле в системе существуют потоки с разными приоритетами, а это меняет порядок распределения процессорного времени.

Каждому потоку присваивается уровень приоритета – от 0 (самый низкий) до 31 (самый высокий). Решая, какому потоку выделить процессорное время, система сначала рассматривает только потоки с приоритетом 31 и подключает их к процессору по принципу карусели. Если поток с приоритетом 31 не исключен из планирования, он получает квант времени, по истечении которого система проверяет, есть ли еще один такой поток. Если есть, то и он получает свой квант процессорного времени.

Пока в системе имеются планируемые потоки с приоритетом 31, ни один поток с более низким приоритетом процессорного времени не получит. Такая ситуация называется голоданием (starvation). Она наблюдается, когда потоки с более высоким приоритетом так интенсивно используют процессор, что остальным ничего не достается.

Кроме того, потоки с более высоким приоритетом вытесняют потоки с более низким приоритетом. Допустим, процессор исполняет поток с приоритетом 5, и тут система обнаруживает, что поток с более высоким приоритетом готов к выполнению. Тогда система остановит поток с более низким приоритетом – даже если не истек отведенный ему квант процессорного времени – подключит к процессору поток с более высоким приоритетом и выдаст ему полный квант времени.

Процессор может выполнять не более одного потока одновременно. Однако *многозадачная* *(multitasking)* ОС дает пользователю возможность исполнять несколько программ, причем создается впечатление, что все они исполняются одновременно. Это достигается следующим образом:

* поток исполняется до тех пор, пока его исполнение не будет прервано или ему не придется ждать освобождения некоторого ресурса;
* cохраняется контекст потока;
* загружается контекст другого потока;
* этот цикл повторяется до тех пор, пока есть потоки, ожидающие выполнения.

Более детально это изображено на Рис. 5 Состояния потоков.



Рис. 5 Состояния потоков

Переключение процессора с исполнения одного потока на исполнение другого потока называется *переключением контекста (context switching).* В Windows NT оно осуществляется ядром.

Как показано на примере двух потоков Рис. 6 Многозадачность, ОС поочередно выполняет то один поток, то другой. В конце концов, каждый поток заканчивает выполнение своей подзадачи и завершается. Очень высокая скорость работы процессора обеспечивает иллюзию одновременного выполнения всех потоков.

Многозадачность увеличивает объем работы, выполняемой системой, потому что большинство потоков не могут исполняться непрерывно. Периодически поток прекращает выполнение и ждет пока другой поток, например, освободит ресурс необходимый первому. Благодаря многозадачности, пока один поток ожидает, может выполняться другой поток, и время процессора не расходуется впустую.



Рис. 6 Многозадачность

*Вытесняющая многозадачность (preemptive multitasking)* – это разновидность многозадачности, при которой ОС не ждет, пока поток добровольно предоставит процессор другим потокам. Вместо этого ОС прерывает поток, после того как он выполнялся в течение заранее заданного периода времени, так называемого *кванта времени (time quantum),* или когда готов к выполнению поток с большим приоритетом. Вытеснение предотвращает монополизацию процессора одним потоком и предоставляет другим потокам их долю процессорного времени. Windows NT – это система с вытесняющей многозадачностью. В невытесняющих системах, поток должен был добровольно передавать управление процессором. Плохие программы могут захватить процессор и нарушить работу других приложений или всей системы.

Во многих ОС программы могут иметь только один поток. Фактически в большинстве ОС для обозначения исполняемой сущности используется слово *процесс (process)*. *Поток (thread) –* новый термин. Так как у каждого процесса имеется отдельное адресное пространство, двум процессам для обмена данными друг с другом нужны специальные средства. Например, для связи между процессами используются каналы (см. Рис. 7 Компилятор, состоящий из двух процессов).



Рис. 7 Компилятор, состоящий из двух процессов

Предпроцессорная обработка и компиляция программы двумя процессами (каждый с одним потоком), будет быстрее, чем в случае одного процесса, так как многозадачная ОС может попеременно исполнять то поток препроцессора, то поток компилятора. Как только препроцессор поместит что-нибудь в совместно используемый буфер, компилятор может начать свою работу. Приложения, которые исполняются в двух или более местах одновременно, называются *параллельными приложениями (concurrent applications).*

ОС мультипроцессорной обработкой специально спроектирована для работы на компьютерах с несколькими процессорами. ОС с *симметричной мультипроцессорной обработкой (symmetric multiprocessing SMP)*, такая как Windows NT, может выполнять на любом процессоре, как пользовательский код, так и код ОС. Если число потоков превышает число процессоров, ОС SMP также поддерживает многозадачность, разделяя время каждого процессора между всеми ожидающими потоками (см. Рис. 8 Мультипроцессорная обработка).



Рис. 8 Мультипроцессорная обработка

### Многопоточность

Использование двух процессов для достижения параллельности не всегда эффективно. В некоторых системах UNIX, когда один процесс создает другой, система должна скопировать все содержимое адресного пространства первого процесса в адресное пространство второго (клонирование). Для большого адресного пространства эта операция занимает много времени. Кроме того, два процесса должны создать канал для обмена данными. Не во всех ОС это осуществляется быстро и легко.

Бывают ситуации, когда более выгоден другой подход к достижению параллельности, а именно, *многопоточный процесс (multithreaded process).* У многопоточного процесса имеется два и более потоков (счетчиков команд), которые совместно используют одно и то же адресное пространство и описатели объектов.

Каждый процесс NT создается с одним потоком. При необходимости можно создать внутри процесса дополнительные потоки. Они часто используются для выполнения в программе *асинхронных операций (asynchronous operations)*, т.е. операций, которые могут иметь место в любое время, безотносительно к основному течению программы. Например, можно использовать поток для периодического сохранения редактируемого документа или опроса устройства ввода, например мыши или клавиатуры. Используя один поток для выполнения основной программы, и создав второй для опроса устройства ввода, система может по отдельности выполнять обе операции.

Чтобы достичь параллелизма с использованием потоков, программа создает два или несколько потоков в одном процессе. Многопоточный компилятор изображен на Рис. 9 Многопоточный компилятор.



Рис. 9 Многопоточный компилятор

Многопоточные процессы добиваются параллельности и при этом не имеют недостатков, связанных с использованием двух процессов. Потоки требуют меньших издержек и создаются быстрее, чем процессы (их называют легковесными процессами). Кроме того, поскольку все потоки процесса используют одну и ту же память, за исключением своих стеков и содержимого регистров, не требуется никакого механизма обмена данными. Один поток просто записывает данные в память, а другой считывает их. Аналогично, все ресурсы процесса (объекты) одинаково доступны всем его потокам.

Для выбора порядка, в котором исполняются потоки, ядро NT использует приоритеты. Потоки с большим приоритетом исполняются прежде потоков с меньшим приоритетом. Ядро периодически изменяет приоритет каждого потока, чтобы гарантировать выполнение их всех.

Использование многопоточного процесса – идеальное решение для серверного приложения (например, защищенных подсистем Windows NT), которое принимает запросы от клиентов и выполняет по каждому запросу один и тот же код. Например, файл-сервер выполняет операции с файлами; он открывает файлы, читает, пишет в них и закрывает их. Хотя каждый запрос может требовать от сервера работы со своим файлом, программа сервера загружается в память только один раз. Каждый приходящий запрос принимается и обрабатывается отдельным потоком сервера. Все запросы клиентов обслуживаются параллельно (см. Рис. 10 Многопоточный сервер).

Писать многопоточное приложение надо очень тщательно, так как все потоки процесса имеют полный доступ к его адресному пространству. Потоки могут пересечься друг с другом, выполняя чтение или запись в память не в том порядке. Если мы используем два процесса, которые взаимодействуют через канал, такого не произойдет. Поэтому защищенные подсистемы реализованы как отдельные процессы и соответственно у каждой из них отдельное адресное пространство и вмешательство других подсистем или пользовательских приложений не допускается.



Рис. 10 Многопоточный сервер

### Объект-поток

Процесс NT остается мертвым без потока, который можно направить на выполнение. Как и процессы, потоки исполнительной системы NT реализованы в виде объектов. Объект – поток изображен Рис. 11 Объект-поток. В исполнительной системе потоки – это объекты исполнительной системы, создаваемые и уничтожаемые диспетчером объектов. Диспетчер процессов определяет атрибуты, хранящиеся в теле объектов потоков, а также предоставляет системные сервисы для чтения и изменения этих атрибутов. Основные атрибуты и сервисы для объектов

потоков приводятся на рисунке. Объект поток исполнительной системы включает объект поток ядра (содержит указатель на объект поток ядра). Ядро управляет объектом поток ядра, а исполнительная система управляет объектом исполнительной системы.

Основные атрибуты этих объектов приводятся по отдельности, для того чтобы лучше понять какие задачи решаются ядром, а какие исполнительной системой, так же как и для процесса.



Рис. 11 Объект-поток

Рассмотрим основные атрибуты:

* *идентификатор клиента* – уникальное значение, идентифицирующее поток;
* *контекст потока –* набор значений регистров, определяющих состояние выполнения потока с учетом специфики конкретного процессора;
* *динамический приоритет –* приоритет планирования потока на данный момент;
* *базовый приоритет* – нижний придел динамического приоритета потока;
* *процессорное сродство потока* – набор процессоров, на которых может исполняться поток, подмножество процессорного сродства процесса потока;
* *время выполнения потока* – общее время выполнения потока в пользовательском режиме и режиме ядра.

Некоторые атрибуты потока напоминают атрибуты процесса. Одни атрибуты, такие как процессорное сродство потока и динамический приоритет, могут усилить или ослабить ограничения, задаваемые для процесса в целом. Например, процессорное сродство каждого потока – это подмножество процессорного сродства процесса. Аналогично, каждый поток имеет базовый приоритет, который лежит в диапазоне от двух уровней ниже базового приоритета процесса до двух уровней выше этого приоритета (см. Рис. 12 Соотношение приоритетов). На рисунке показан также динамический приоритет потока, нижней границей которого является базовый приоритет потока, а верхняя граница зависит от вида работ, исполняемых потоком.



Рис. 12 Соотношение приоритетов

### Планирование потоков

Теперь рассмотрим планирование коротко и более формально

В Windows реализовано вытесняющее планирование на основе уровней приоритета, при котором всегда выполняется поток с наибольшем приоритетом, готовый к выполнению. При выборе потока также учитывается привязка к процессорам.

Выбранный для выполнения поток работает в течение некоторого периода времени, называемого квантом. Квант определяет, сколько времени будет работать поток, пока ОС не прервет его. После этого ОС будет решать, искать другой готовый к выполнению поток с тем же приоритетом или снизить приоритет данного потока.

Поток может не полностью использовать свой квант. Если появляется поток с более высоким приоритетом готовый к выполнению, то текущий поток вытесняется, даже если не истек его квант.

Код, отвечающий за планирование, реализован в ядре. Набор процедур, которые этим занимаются, называются ***диспетчером ядра*** (***kernel’s dispatcher***). Диспетчеризация потоков инициализируется любым из следующих событий:

* поток готов к выполнению (он создан или вышел из состояния ожидания);
* поток вышел из состояния выполнения (истек его квант, он завершился или перешел в состояние ожидания);
* приоритет потока изменился;
* изменилась привязка к процессорам.

#### Квант

Квант – это интервал процессорного времени, отведенный потоку для выполнения. По истечении кванта ОС проверяет готов ли к выполнению другой поток с таким же приоритетом. Если такого нет, то ОС выделяет текущему потоку еще один квант.

Потокам выделяются разные кванты в зависимости от ОС (Windows 2000 Professional или Windows 2000 Server). По умолчанию начальная величина кванта в Windows 2000 Professional равна 6, а в Windows 2000 Server – 36.

Windows может динамически увеличивать величину кванта. Поясним это на примере. Например, вы начали пересчет большой таблицы и запустили игру. Если повысить приоритет игры, то фоновый процесс пересчета будет получать очень маленькую часть процессорного времени. А увеличение кванта для игры не будет блокировать пересчет, просто игре будет отдано предпочтение.

#### Сценарии планирования

##### Самостоятельное переключение

Поток может самостоятельно освободить процессор, перейдя в состояние ожидания на каком-либо объекте (***WaitForSingleObject*** или ***WaitForMultipleObjects***).

##### Вытеснение

Поток с более низким приоритетом вытесняется готовым к волнению потоком с более высоким приоритетом. Такая ситуация может возникнуть в двух случаях:

* поток с более высоким приоритетом “дождался” (произошло событие, которого он ждал);
* приоритет потока увеличился или уменьшился.

##### Завершение кванта

Поток израсходовал свой квант процессорного времени, и ОС должна решить, следует ли понизить его приоритет и подключить к процессору другой поток.

##### Завершение потока

Завершаясь (после возврата из основной процедуры и вызова ***ExitThread*** или ***TerminateThread***) поток переходит в состояние Terminated.

#### Динамическое повышение приоритета

Динамическое повышение приоритета предназначено для оптимизации общей пропускной способности и отзывчивости системы, а также для устранения потенциально нечестных сценариев планирования. Но это не панацея и от него выигрывают не все приложения.

Windows может динамически повышать значение текущего приоритета потока в одном из следующих случаев:

* после завершения операции ввода/вывода;
* по окончании ожидания на каком-либо объекте исполнительной системы;
* при нехватке процессорного времени.

В первом случае ОС временно повышает приоритет потоков по окончании операций ввода/вывода. В этом случае у таких потоков появляется больше шансов возобновить выполнение и обработать полученные данные. После динамического повышения приоритета поток в течение одного кванта выполняется с этим приоритетом, после чего приоритет уменьшается на один уровень и потоку выделяется еще один квант. Этот цикл продолжается до тех пор пока приоритет не снизится до базового.

Во втором случае (***SetEvent, ReleaseSemaphore***) приоритет потока увеличивается на один уровень.

В третьем случае рассмотрим следующий сценарий. Поток с приоритетом 7 не дает выполняться потоку с приоритетом 4, а при этом поток с приоритетом 11 ожидает ресурс, заблокированный потоком с приоритетом 4. В подобной ситуации диспетчер настройки баланса сканирует очереди готовых потов и ищет потоки, которые находились в состоянии Ready более 3 секунд. Обнаружив такой поток, диспетчер повышает его приоритет до 15 и выделяет ему квант вдвое больше обычного. По истечении двух квантов приоритет потока снижается до исходного уровня.

#### Абстрагирование приоритетов

Windows поддерживает шесть классов приоритета процесса: idle (простаивающий), below normal (ниже обычного), normal (обычный), above normal (выше обычного), high (высокий), real-time (реального времени).

***Real-time*** - это наивысший возможный приоритет. Потоки в этом процессе обязаны немедленно реагировать на события. Исполнение таких потоков может привести к полной блокировке системы. Будьте очень осторожны с этим классом.

***High*** – потоки в этом процессе тоже должны немедленно реагировать на события ( этот класс присвоен Task Manager ).

***Above normal*** – класс приоритета промежуточный между normal и high. Новый класс введенный в Windows 2000.

***Normal –*** потоки в этом процессе не предъявляют особых требований к выделению им процессорного времени.

***Below normal*** - класс приоритета промежуточный между normal и idle. Новый класс введенный в Windows 2000.

***Idle*** – потоки в этом процессе выполняются, когда система не занята другой работой. Этот класс приоритета обычно используется для утилит, работающих в фоновом режиме.

Кроме того, Windows поддерживает семь относительных приоритетов потоков: idle (простаивающий), lowest (низший), below normal (ниже обычного), normal (обычный), above normal (выше обычного), highest (высший) и time-critical (критичный по времени). Эти приоритеты относительны классу приоритета процесса. Относительные приоритеты потоков описаны в таблице (см. Таблица 1 Относительные приоритеты потоков).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Класс приоритета процесса | | | | | | |
| Относительный приоритет потока | Idle | Below normal | Normal | Above normal | High | Real-time |
| Time-critical | 15 | 15 | 15 | 15 | 15 | 31 |
| Highest | 6 | 8 | 10 | 12 | 15 | 26 |
| Above normal | 5 | 7 | 9 | 11 | 14 | 25 |
| Normal | 4 | 6 | 8 | 10 | 13 | 24 |
| Below normal | 3 | 5 | 7 | 9 | 12 | 23 |
| Lowest | 2 | 4 | 6 | 8 | 11 | 22 |
| Idle | 1 | 1 | 1 | 1 | 1 | 16 |

Таблица Относительные приоритеты потоков

## Лабораторная работа 2 Процессы и потоки (программирование)

### Теория

В этой лабораторной работе представлена методология создания ***параллельных приложений*** (***concurrent application***). Такое приложение может исполняться в двух и более местах одновременно. Написание параллельных приложений полезно по двум причинам. Во-первых, современное ПО разрабатывается по частям, которые могут выполняться параллельно. Во-вторых, когда приложение использует параллельную модель, современные ОС, такие как Windows NT, представляют много возможностей для разработчиков для управления параллельными приложениями. В этой лабораторной работе параллельность рассматривается на уровне процесса и потока. Обычные программы, которые вы писали, выполнялись как один процесс с одним потоком (последовательное программирование). Современное поколение ПО использует преимущества многих процессоров на вашем компьютере или доступных по сети. Таким образом, традиционные последовательные модели вычислений заменяются параллельными моделями со многими процессами и потоками. В лабораторной работе будет представлена информация о процессах и потоках Windows NT и рассмотрено создание процессов и потоков, используя Win32 API.

#### Создание процесса

Один процесс может создать другой, вызывая Win32 API ***CreateProcess*** (при этом вызове используются Native API *NTCreateProcess* и *NTCreate Thread*). Когда создается процесс исполнительная система (Executive) выполняет большой объем работы. Она выделяет новое адресное пространство и ресурсы для процесса, а также создает для него новый базовый поток. Когда новый процесс создан, старый процесс будет продолжать исполняться, используя старое адресное пространство, а новый будет выполняться в новом адресном пространстве с новым базовым потоком. Существует много различных опций для создания процесса, поэтому функция ***CreateProcess*** имеет много параметров, причем некоторые из них достаточно сложные. После того, как исполнительная система создала новый процесс, она возвращает его описатель (handle), а также описатель его базового потока. Рассмотрим прототип функции ***CreateProcess***. В прототипе не используются стандартные типы С, а вместо этого используется набор типов, определенный в **windows.h**.

BOOL ***CreateProcess***(

LPCTSTR lpApplicationName,

// имя исполняемого модуля (указатель)

LPTSTR lpCommandLine,

// командная строка (указатель)

LPSECURITY\_ATTRIBUTES lpProcessAttributes,

// атрибуты безопасности процесса (указатель)

LPSECURITY\_ATTRIBUTES lpThreadAttributes,

// атрибуты безопасности потока (указатель)

BOOL bInheritHandeles, // флаг наследования описателя

DWORD dwCreationFlags, // флаги создания

LPVOID lpEnvironment,

// новый блок окружения (указатель)

LPCTSTR lpCurrentDirectory,

// имя текущей директории (указатель)

LPSTARTUPINFO lpStartupInfo,

// STARTUPINFO (указатель)

LPPROCESS\_INFORMATION lpProcessInformation

// PROCESS\_INFORMATION (указатель)

)

Десять параметров CreateProcess обеспечивают большую гибкость в использовании для программиста, хотя в простейшем случае для многих параметров можно использовать значения по умолчанию (by default). Здесь мы рассмотрим относительно простой набор параметров, а более детально можно посмотреть в Microsoft Developers Network (MSDN).

##### Параметры lpApplicationName и lpCommandLine

Два первых параметра обеспечивают два различных способа определения имени файла, который будет выполняться базовым потоком процесса. lpApplicationName – эта строка содержит имя файла, который будет выполняться, а lpCommandLine – эта строка содержит командную строку для запуска процесса в cmd.exe. Существует набор правил, определяющий в каком случае какое имя использовать (см. MSDN).

Параметр lpApplicationName определяет исполняемый модуль. В строке, на которую указывает этот указатель, надо задать:

* полный путь доступа и имя файла, включая расширение (**.exe** или **.bat**)
* только имя тогда будет использоваться текущий диск и каталог.

Если значение параметра lpApplicationName равно NULL, то именем исполняемого модуля должна быть первая из разделенных пробелами лексем, заданная параметром lpCommandLine. Если полный путь доступа не задан, то поиск файла будет производится в следующем порядке:

* текущий каталог
* системный каталог Windows (GetSystemDirectory)
* каталог Windows (GetWindowsDirectory)
* каталоги, заданные в переменной окружения PATH.

Будем задавать NULL для lpApplicationName и командную строку для lpCommandLine. Предположим, вы хотите создать процесс для запуска notepad.exe с файлом temp.txt.

##### Параметры lpProcessAttributes, lpThreadAttributes и bInheritHandeles

В данном случае будем использовать значения по умолчанию для атрибутов безопасности процесса и потока – NULL и FALSE для флага наследования (атрибуты безопасности и флаг наследования будут рассмотрены далее).

***CreateProcess***(NULL, lpCommandLine, NULL, NULL, FALSE, …);

##### Параметр DwCreationFlags

Этот параметр используется для управления приоритетом нового процесса и другими особенностями.

Порожденный процесс - процесс-ребенок (child) может быть создан с одним из четырех классов приоритетов HIGH\_PRIORITY\_CLASS, IDLE\_PRIORITY\_CLASS, NORMAL\_PRIORITY\_CLASS или REALTIME\_PRIORITY\_CLASS. Значение по умолчанию - NORMAL\_PRIORITY\_CLASS, но если порождающий процесс (процесс родитель parent) имеет приоритет IDLE\_PRIORITY\_CLASS, то и процесс-ребенок также будет иметь приоритет IDLE\_PRIORITY\_CLASS.

Потоки процесса с приоритетом HIGH\_PRIORITY\_CLASS будут вытеснять потоки процессов с приоритетами IDLE\_PRIORITY\_CLASS или NORMAL\_PRIORITY\_CLASS. Потоки процесса с приоритетом IDLE\_PRIORITY\_CLASS будут выполняться только в том случае, если нет других потоков для выполнения.

Потоки процесса с приоритетом REAL\_PRIORITY\_CLASS будут вытеснять потоки всех других классов.

Можно задать и другие флаги, а также использовать вместе (все возможные значения см. в MSDN). Здесь нам будет полезен флаг CREATE\_NEW\_CONSOLE – новый процесс будет создан в своем собственном окне cmd.exe.

Для создания нового процесса (child) с высоким приоритетом в его собственном окне используйте - HIGH\_PRIORITY\_CLASS | CREATE\_NEW\_CONSOLE.

***CreateProcess***(NULL, lpCommandLine, NULL, NULL, FALSE,

HIGH\_PRIORITY\_CLASS | CREATE\_NEW\_CONSOLE, …);

##### Параметр lpEnvironment

Используется для передачи нового блока переменных окружения порожденному процессу-ребенку (child). Если NULL, то ребенок использует то же окружение, что и родитель. Если не NULL, то lpEnvironment должен указывать на массив строк, каждая name=value. Здесь мы будем использовать NULL.

***CreateProcess***(NULL, lpCommandLine, NULL, NULL, FALSE,

HIGH\_PRIORITY\_CLASS | CREATE\_NEW\_CONSOLE, NULL, …);

##### Параметр lpCurrentDirectory

Указатель на строку, содержащую путь к текущему каталогу нового процесса. Если использовать NULL, то в качестве текущего каталога будет использован каталог родителя.

***CreateProcess***(NULL, lpCommandLine, NULL, NULL, FALSE,

HIGH\_PRIORITY\_CLASS | CREATE\_NEW\_CONSOLE, NULL, NULL, …);

##### Параметр lpStartupInfo

Это указатель на следующую структуру, которая определяет внешний вид окна и содержит дескрипторы стандартных устройств ввода/вывода/ошибки нового процесса:

typedef struct\_STARTUPINFO { //si

DWORD cb; // длина структуры

LPTSTR lpReserved;

LPTSTR lpDesktop;

LPTSTR lpTitle;

DWORD dwX;

DWORD dwY;

DWORD dwXSize;

DWORD dwSizeY;

DWORD dwXCountChars;

DWORD dwYCountChars;

DWORD dwFillAttribute;

DWORD dwFlags;

DWORD dwShowWindow;

WORD cbReserved2;

LPBYTE lpReserved2;

HANDLE hStdInput;

HANDLE hStdOutput;

HANDLE hStdError; } STARTUPINFO, \*LPSTARTUPINFO;

Экземпляр этой структуры данных должен быть создан в вызывающей программе (описание полей структуры см. в MSDN). Затем ее адрес должен быть передан как параметр в CreateProcess.

Внимание!!! CreateProcess не знает длины структуры, поэтому обязательно инициализируйте поле размера в STARTUPINFO (поле .cb) до того как передать указатель.

STARTUPINFO startupInfo;

…

ZeroMemory(&startupInfo,sizeof(STARTUPINFO));

startupInfo.cb=sizeof(startupInfo);

***CreateProcess***(NULL, lpCommandLine, NULL, NULL, FALSE,

HIGH\_PRIORITY\_CLASS | CREATE\_NEW\_CONSOLE, NULL, NULL, &startupInfo, …);

##### Параметр lpProcessInformation

Это указатель на структуру, в которую будут помещены возвращаемые функцией значения дескрипторов и глобальных идентификаторов процесса и потока:

Typedef struct\_PROCESS\_INFORMATION {

HANDLE hProcess;

HANDLE hThread;

DWORD dwProcessId;

DWORD dwThreadId; } PROCESS\_INFORMATION;

В этой структуре нет поля длины, поэтому процессу нужно только создать экземпляр PROCESS\_INFORMATION и передать указатель на него функции CreateProcess. После возврата, поля структуры будут заполнены, и вы получите:

* Дескриптор (описатель) вновь созданного процесса (hProcess)
* Дескриптор (описатель) его базового потока (hThread)
* Глобальный идентификатор процесса (dwProcessId)
* Глобальный идентификатор потока (dwThreadId).

CreateProcess отводит место под объекты процесс и поток и возвращает значения их описателей (индексы в таблице) в структуре PROCESS\_INFORMATION. Вы можете освободить выделенное место, используя CloseHandle. Конечно, когда процесс заканчивается, все его описатели автоматически освобождаются. Хотя описатель это системный ресурс, хорошая практика, если вы будете закрывать описатель, если не используете его.

CreateProcess возвращает ноль, если все плохо и не ноль в противном случае.

Далее приводится код для создания процесса:

#include <windows.h>

#include <stdio.h>

#include <string.h>

…

STARTUPINFO startInfo;

PROCESS\_INFORMATION processInfo;

…

strcpy(lpCommandLine,“C:\\WINNT\\SYSTEM32\\NOTEPAD.EXE temp.txt”);

ZeroMemory(&startupInfo,sizeof(STARTUPINFO));

startupInfo.cb=sizeof(startupInfo);

if(!***CreateProcess***(NULL, lpCommandLine, NULL, NULL, FALSE,

HIGH\_PRIORITY\_CLASS | CREATE\_NEW\_CONSOLE, NULL, NULL, &startupInfo, &processInfo))

{

fprintf(stderr,”CreateProcess failed on error %d\n”,GetLastError());

ExitProcess(1);

}

…

CloseHandle(processInfo.hThread);

CloseHandle(processInfo.hProcess);

#### Создание потока

Вы можете создать дополнительные потоки в текущем процессе, используя функцию ***CreateThread*** Win32 API (которая использует *NTCreateThread* Native API). Каждый поток это отдельная исполняемая сущность внутри адресного пространства процесса. Для создания потока программист должен задать необходимую информацию.

Рассмотрим прототип функции:

HANDLE ***CreateThread***(

LPSECURITY\_ATTRIBUTES lpThreadAttributes,

// атрибуты безопасности потока (указатель)

DWORD dwStackSize,

// начальный размер стека потока в байтах

LPTHREAD\_START\_ROUTINE lpStartAddress,

// функция потока (указатель)

LPVOID lpParameter, // аргумент для нового потока

DWORD dwCreationFlags, // флаги создания

LPDWORD lpThreadId

// возвращаемый идентификатор потока (указатель)

);

Этот прототип использует шесть параметров для описания характеристик нового потока. Функция создает описатель потока, и возвращать его как результат вызова. Это означает, что системные ресурсы размещаются при удачном вызове ***CreateThread*** и программист должен закрыть описатель, когда он больше не используется.

##### Параметр lpThreadAttributes

Это то же самое, что и атрибуты безопасности в CreateProcess только для потока. В простейшем случае NULL (будет рассмотрено далее).

***CreateThread***(NULL, …);

##### Параметр dwStackSize

Каждый поток выполняется независимо от других потоков процесса и имеет свой собственный стек. Используя этот параметр, программист может установить размер стека. Обычно используется значение по умолчанию – 0.

***CreateThread***(NULL, 0, …);

##### Параметры lpStartAddress и lpParameter

Для процесса необходимо было задать имя выполняемого файла. Для потока необходимо задать адрес в текущем адресном пространстве, где новый поток должен выполняться. LpStartAddress – это и есть такой адрес. В языках программирования (таких как С) обычно невозможно задать в качестве стартового адреса адрес в середине процедуры. Нужно использовать точку входа в процедуру (entry point). LpStartAddress – это точка входа для функции, которая имеет следующий прототип:

DWORD WINAPI ThreadFunc(LPVOID);

Как передать параметр функции, которая будет выполняться новым потоком? Необходимо использовать lpParameter, он будет передан функции как параметр, когда новый поток начнет ее выполнение.

Пусть дан прототип функции, которую будет выполнять новый поток:

DWORD WINAPI myFunc(LPVOID);

Предположим, что родительский поток хочет передать целый аргумент новому детскому потоку:

int theArg;

…

***CreateThread***(NULL, 0, myFunc, &theArg, …);

##### Параметр dwCreationFlags

Этот параметр используется для определения, каким образом создать новый поток. Возможное значение CREATE\_SUSPENDED – в этом случае новый поток будет создан, но приостановлен до тех пор, пока другой поток не выполнит вызов:

ResumeThread(targetThreadHandle);

Где targetThreadHandle описатель нового потока. Значение по умолчанию 0 – в этом случае поток будет активизирован сразу после создания.

***CreateThread***(NULL, 0, myFunc, &theArg, 0, …);

##### Параметр lpThreadId

Это указатель на идентификатор потока (аналог поля dwThreadId в структуре PROCESS\_INFORMATION, которую возвращает CreateProcess).

DWORD targetThreadId;

…

***CreateThread***(NULL, 0, myFunc, &theArg, 0, & targetThreadId);

### Чему нужно научиться

Интерпретатор командной строки (cmd.exe в NT, command.com в DOS или один из shell UNIX) это программа, которая обеспечивает текстовый интерфейс для общения пользователя с ОС. Пользователь запускает программу из командной строки интерпретатора, печатая имя файла, который содержит выполняемую программу, с набором параметров. Это таже самая информация, которая задается параметром lpCommandLine в CreateProcess. Интерпретатор разбирает строку, чтобы получить имя файла, а затем заставляет процесс выполнить команду. В UNIX shell процесс командной строки действительно создает новый процесс, который загрузит и выполнит программу из файла. Интерпретатор командной строки ждет, пока команда выполнится, и ее поток закончится, перед тем как предоставить пользователю возможность дальше вводить команды.

Нужно научиться писать программы на С, используя функции ***CreateProcess*** и ***CreateThread*** для создания:

* одного или нескольких процессов (каждый с базовым потоком) – задание **А;**
* нескольких потоков в одном процессе – задание **В**.

### Задания

#### Уровень 1 (A)

Используя функцию ***CreateProcess***, создайте процесс для запуска notepad.exe.

#### Уровень 2 (A)

Используйте текстовый редактор (например, notepad) для подготовки конфигурационного файла. Он может содержать следующий перечень:

С:\\WINNT\\SYSTEM32\\NOTEPAD.EXE our.txt

С:\\WINNT\\SYSTEM32\\CALC.EXE

С:\\WINNT\\SYSTEM32\\CHARMAP.EXE

Напишите программу, которая получает имя конфигурационного файла из командной строки, должна открыть конфигурационный файл, прочитать строки и создать процесс для запуска каждой команды.

#### Уровень 3 (A)

Необходимо выполнить задание Уровень 3 (В).

#### Уровень 1 (В)

**Важно!!!** Нужно использовать /MTd вместо /MLd, чтобы указать компилятору, что код будет многопоточным.

Создайте два дополнительных потока в вашем процессе. Пусть функция вашего первого потока в бесконечном цикле выводит сообщение (например, “Я поток 1 и я выполняюсь!”), а функция вашего второго потока в бесконечном цикле выводит сообщение (например, “Я поток 2 и я выполняюсь!”). Базовый поток после создания этих двух дополнительных потоков пусть тоже в бесконечном цикле выводит сообщение (например, “Я базовый поток и я выполняюсь!”). Добейтесь того, чтобы каждый из трех потоков поочередности выводил на экран эти слова.

Затем измените программу так, чтобы после создания двух дополнительных потоков, базовый поток заканчивал работу. Объясните полученный вывод.

Теперь измените программу так, чтобы после создания двух дополнительных потоков, базовый поток вызывал функцию ***Sleep***. Объясните полученный вывод.

#### Уровень 2 (В)

Создать в цикле несколько дополнительных потоков в процессе. В качестве аргументов командной строки задать число создаваемых потоков и время их существования. Пусть функция потока в бесконечном цикле выводит сообщение (например, “Я поток с номером N и я выполняюсь!”).

Используйте системное время. Время, которое процесс и потоки в нем должны существовать, задается как параметр, и по истечении этого времени сообщество должно погибнуть. После того как рабочие потоки созданы и выполняют свою работу, поток координатор проверяет текущее время, чтобы определить, не наступил ли назначенный час. Если нет, то он засыпает на 1сек, а потом просыпается и проверяет время снова. Когда время истечет поток координатор устанавливает глобальную переменную runFlag=FALSE, ждет 5сек и заканчивается. Используйте разделяемую всеми потоками глобальную переменную runFlag.

#### Уровень 3 (В)

Разработайте программу, которая позволит изменять класс приоритета процесса и приоритеты потов этого процесса. Как значение приоритета влияет на выделение процессорного времени? Что будет, если запретить динамическое изменение приоритета?

Изучите и используйте в своей программе функции: Изучите и используйте в своей программе функции: ***GetPriorityClass, SetPriorityClass, SetProcessPriorityBoost, GetProcessPriorityBoost, ExitProcess, TerminateProcess***, ***GetThreadPriority, SetThreadPriority, SetThreadPriorityBoost, ThreadPriorityBoost, SuspendThread и ResumeThread.***