**Есть два типа сообщений: системные и прикладные.**

**Системные сообщения отправляются системой при взаимодействии с приложением. Эти сообщения используются для управления приложением, также предоставляют ввод и другую информацию для приложений.**

**Системные сообщения может отправлять не только ws, но и приложение.**

**Приложения обычно используют системные сообщения для управления окном через класс окна.**

**Прикладные сообщения: приложение может использовать сообщения для своих собственных окон или для взаимодействия с окнами, принадлежащими другим процессам.**

**Если приложение порождает своё сообщение, процедура окна, получающего это сообщение, должна его интерпретировать и выполнять какие-то действия.**

**Так как сообщение это самое обычное целое число типа short, их также разделяют на диапазоны в зависимости от их специфики (области применения).**

**Диапазон от 0x0000 до 0х03FF - сообщения, определённые системой. Эти сообщения не могут быть использованы приложениями.**

**Сообщения от 0x0400 до 0х7FFF доступен для приватных классов окон, то есть для приложений.**

**С версии 4 можно использовать сообщения от 0x8000 до 0хBFFF.**

**Четвёртый диапазон: система возвращает сообщения в диапазоне от 0xC000 до 0хFFFF, когда приложение использует функцию .**

**Идентификатор сообщения, возвращаемый этой функцией, будет гарантированно уникален в системе.**

**Использование этой функции предотвращает конфликты, которые могут возникать, когда другие приложения используют сообщения с таким же номером для других целей.**

**Для перевода любого числа в десятичную систему исчисления необходимо каждый разряд умножить на весовой коэффициент этого разряда.**

**Разрядность и величина числа имеют прямую зависимость.**

**В позиционных системах исчисления любая цифра представлена символом.**

**Маршрутизация сообщений**.

Система использует два метода для отправки сообщений в процедуру окна.

Первый метод – система помещает сообщение в очередь сообщений.

Очередь сообщений – обычный FIFO.

Очередь сообщений – блок памяти, выделенный системой, в котором временно хранятся сообщения, и через который они отправляются непосредственно в процедуру окна.

Сообщения, передаваемые через очередь сообщений, называют queuered messages (qm).

В очередь сообщений помещяются сообщений от клавиатуры и мыши, 

Большинство других сообщений на ставятся в очередь (non-queuered messages (NQM).



Система может одновременно отображать любое количество окон. Для маршрутизации ввода с клавиатуры и мыши в соответствующее окно система использует очереди сообщений.

В системе одновременно существует одна системная очередь сообщений, и у каждого потока, использующего GUI (Graphical User Interface), также есть своя отдельная очередь сообщений.

Чтобы избежать пересечения очередей для потоков без графического интерфейса, все потоки Очередь сообщений для потока создается только в том случае, когда поток вызывает пользовательскую функцию.

Когда пользователь перемещает мышь, нажимает кнопки мыши или клавиши на клавиатуре, драйвер клавиатуры и мыши преобразует ввод в сообщение и помещает его в системную очередь сообщений.

Система удаляет сообщение из системной очереди и одновременно с этим определяет окно, к которому адресовано это сообщение, и помещает удаляемое сообщение в очередь нужного окна, точнее того потока, который создан окно.

Очередь сообщений потока принимает все сообщения от клавиатуры и мыши для всех окно, созданных этим потоком.

Поток удаляет сообщение из своей очереди и просит систему отправить его в соответствующую процедуру окна.

Система всегда помещает сообщения в конец очереди.

Кроме .

Это гарантирует, что окно получит все сообщения ввода в том порядке, в котором они поступили.

Сообщение  остаются в очереди до тех пор, пока очередь не очистится от других сообщений.

Кроме того, этот экземпляр содержит дескриптор окна, которому адресовано сообщение (hwnd), идентификатор сообщения (двухбайтовое целое число) и два параметра сообщения.

Также экземпляр структуры MSG содержит время загрузки сообщения в очередь и пощицию указателя мыши.

Поток может поместить сообщение в свою собственную очередь иои в очередь другого потока при помощи функции .

Приложение может удалить сообщение из своей очереди при помощи функции .

Для чтения сообщения без удаления из очереди используется функция 

Она заполняет экземпляр структуры msg.

После удаления сообщения из очереди приложение может использовать функцию , чтобы попросить систему отправить его в соответствующую процедуру окна для обработки.

 принимает указатель на экземпляр структуры msg, заполненной предварительным вызовом  или .

Функция  передаёт дескриптор окна, id сообщения и два параметра сообщения в процедуру окна (окно, сообщение, параметры сообщения), но не передаёт время загрузки сообщения в очередь и координаты указателя мыши.

Эту информация приложение может получить при помощи функции  и  при обработке сообщения.

Поток может использовать функцию , чтобы получить управление другими потоками, когда в его очереди нет сообщений.

Функция приостанавливает поток и ничего не возвращает, пока в очередь сообщений не будет что-то загружено.

 может ассоциировать значение с очередью сообщений текущего потока.

Функция  позволяет прочитать это значение.



Эти сообщения немедленно отправляются в целевую процедуру окна, пропуская системную очередь сообщений и очередь сообщений потока, которому принадлежит окно.

Система отправляет такие сообщения, чтобы уведомить окно о некоторых событиях, которые могут повлиять на это окно.

Например, когда пользователь активирует окно приложения, система отправляет ему последовательность сообщений, включая .

Эти сообщения уведомляют окно о том, что оно было активировано, ввод с клавиатуры теперь перенаправляется в это окно, и указатель мыши двигается в пределах этого окна.

NQN могут быть результатом после того, как приложение вызвало некоторые системные функции.

Например, после вызова SetWindowPos, которая задаёт положение окна, в окно прилетает сообщение .

NQN могут быть отправлены функциями  .

**Обработка сообщений.**

Приложение должно обрабатывать сообщения, помещенные в очередь сообщений его потока, и удалять их оттуда.

Однопоточные приложения обычно используют цикл сообщений у функции WnMain для удаления сообщения из очереди и отправку его в процедуру окна для обработки.

Многопоточные приложения могут включать цикл сообщений в каждый поток, порождающий окна.

Рассмотрим более детально цикл сообщений и роль процедуры окна.

**Цикл сообщений (message loop).**

**Простой цикл сообщений при каждой операции делает по одному вызову следующих функций:**

**.**

**В случае возникновения ошибок GetMessage возвращает - 1. Благодаря этому, ошибки можно обрабатывать.**

**Функция GetMessage получает сообщение из очереди и копирует его в структуру msg.**

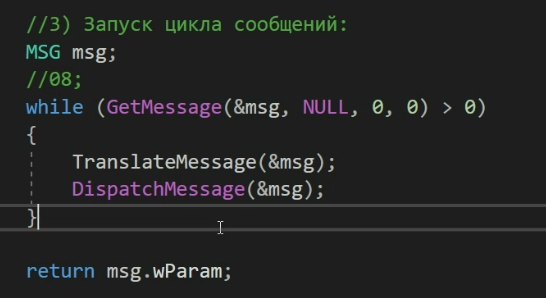
**GetMessage всегда возвращает ненулевое значение, если получает сообщение не** **.**

**В случае с**  **GetMessage возвращает FALSE (0) и цикл сообщений завершается.**

**В однопоточных приложениях это первый шаг к закрытию программы.**

**Приложение может завершить свой цикл при помощи функции** **.**

**Эта функция обычно вызывается в сообщении**  **в процедуре главного окна приложения.**



**Очередь сообщений потока должна вызывать функцию** **, если поток получает ввод с клавиатуры.**

**Система генерирует сообщений виртуальных клавиш** **всякий раз, когда пользователь нажимает на клавиши.**

**Эти сообщения содержат виртуальный код клавиши, определяющий нажатую клавишу, но не определяет символ нажатой клавиши.**

**Чтобы его получить, нужно вызывать функцию**  **в цикле сообщений, которая транслирует сообщения виртуальной клавиши** **.**

**Wm char может быть удалён из очереди последующей итерацией цикла сообщений и отправлен в процедуру окна.**

**Функция**  **отправляет сообщения в процедуру окна, соответствующую обработчику окна (hwnd), указанного в структуре msg.**

**Если дескриптор окна** , **отправляет сообщение процедурам всех главных окон системы.**

**Если hwnd = NULL,** **ничего не делает.**

**Главный поток приложения запускает цикл сообщений после инициализации приложения и создания хотя бы одного окна.**

**После запуска цикл сообщений получает сообщений из очереди его потока и отправляет эти сообщения в соответствующие окна.**

**Цикл сообщений завершается, когда функция GetMessage удаляет из очереди сообщение wm\_quit.**

**Даже если приложение содержит множество окон, достаточно всего лишь одного цикла сообщений, так как**  **всегда отправляет сообщение в нужное окно, так как целевое окно всегда содержится в сообщении.**

**Цикл сообщений можно написать по-разному.**

**Например, сообщения можно не вызывать в окна, то есть не вызывать функцию** **.**

**Это может быть полезно для приложений, которые помещают сообщения в очередь без указания окна.**

**Также GetMessage может доставать из очереди лишь определённые сообщения, игнорируя все остальные сообщения.**

**Если сообщение игнорируется, оно остаётся в очереди.**

**Это может быть полезно, когда нужно временно организовать очередь с приоритетами.**

**Приложения, в которых используются сочетания клавиш, должны иметь возможность транслировать клавиатурные сообщения в команды (wm\_command).**

**Для этого в цикле сообщений должна вызываться функция** **.**

**Если поток использует немодальные окна диалога (не блокирует окно), цикл сообщений должен вызывать функцию** 

**Процедура окна.**

**Это самая обычная функция, которая принимает и обрабатывает все сообщения, отправленные в окно.**

**Абсолютно у каждого класса окна есть своя процедура окна.**

**Следовательно, все окна одного класса обрабатывает одна процедура.**

**Система отправляет сообщение процедуре окна, передавая данные об этом сообщении как параметры в процедуру окна.**

**Процедура окна может обрабатывать или не обрабатывать это сообщение в зависимости от того, как она была написана.**

**Процедура окна обычно не игнорирует сообщение.**

**Даже если она его не обрабатывает, она отправляет его обратно в систему, и сообщение обрабатывается по умолчанию.**

**Для этого процедура окна вызывает функцию** **, которая выполняет некое действие по умолчанию и возвращает результат сообщения, который процедура окна в свою очередь должна вернуть, как свой собственный результат сообщения.**

**Большинство оконных процедур обрабатывает всего несколько сообщений, а все остальные сообщения отправляет обратно в систему функцией** **.**

**Так как одна и та же процедура может обрабатывать сообщения множества окон, их нужно как-то отличать. Именно для этого первым параметром процедура окна принимает окно.**

**Фильтрация сообщений.**

Приложение может извлекать только определённые сообщения из очереди сообщений, игнорируя при этом остальные сообщения.

Такую возможность обеспечивает функция .

Для этого в обеих функциях есть фильтры.

Фильтр – самый обычный диапазон значений, в котором находятся интересующие нас сообщения.

В этом диапазоне можно указать только один передел: только нижний или только верхний.

Фильтрация сообщений может быть полезна, если приложение должно находить сообщение, поступившее в очередь с запозданием, или же когда приложение должно обрабатывать аппаратные сообщения ввода, прежде чем обработать рядовые сообщения в очереди.

Например, константы  позволяют фильтровать сообщения только от клавиатуры,  – только от мыши.

Приложения, фильтрующие сообщение, должны убедиться в том, что сообщение удовлетворяет заданный фильтр.

Например, если приложение фильтрует сообщения , в окне, которое не получает клавиатурный ввод, функция  не вернёт значение, что повесит приложение.



Любое приложение может писать и отравлять сообщения.

Как и система, приложение размещает (post) сообщение, копируя его в очередь сообщений, и отправляет сообщение, передавая его данные как аргументы в процедуру окна.

Для помещения сообщения в очередь приложение использует функцию .

Для отправки сообщения используются функции .



Приложение обычно размещает сообщение в очереди, чтобы уведомить какое-то окно о том, что оно должно что-то выполнить.

 создаёт экземпляр структуры msg для сообщения и копирует его в очередь сообщений.

Цикл сообщений достаёт это сообщение из очереди и отправляет в соответствующую процедуру окна.

Приложение может разместить сообщение в очереди без указания окна, когда сообщение будет размещено в очереди текущего потока.

Так как целевое окно не указано, приложение должно обработать сообщение прямо в цикле.

Это один из способов применить сообщение ко всему приложению вместо определённого окна.

Либо же так можно отправить сообщение всем главным окнам в системе.

Общепринятой ошибкой программирования – мнение о том, что  всегда размещает сообщение в очереди. Это не так, потому что очередь сообщений может быть заполненной.

Поэтому всегда нужно проверять возвращаемое кольцо функции .

Сообщение нужно постить до тех пор, пока оно гарантированно не окажется в очереди, пока  не вернёт ненулевое значение.

Sending Messages

Приложение отправляет сообщение, чтобы процедура окна немедленно завершила задачу. Функция **[SendMessage](https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-sendmessage)** отправляет сообщение процедуре окна, соответствующей заданному окну. Функция ожидает завершение процесса процедурой окна и возвращает сообщение результата. Родительское о дочернее окно часто взаимодействуют, отправляя другу другу сообщения. Например, родительское окно, дочерним окном которого является элемент управления edit (изменить), может задать текст элемента управления, отправив ему сообщение. Такой вид контроля через сообщения может предупредить родительское окно об изменениях текста пользователем.

Функция **[SendMessageCallback](https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-sendmessagecallbacka)** отправляет сообщение процедуре окна, соответствующей заданному окну. Но данная функция немедленно возвращается. После передачи сообщения процедурой окна система обращается к специальной функции обратного вызова. Для большей информации рассмотрим функцию **[SendAsyncProc](https://learn.microsoft.com/en-us/windows/win32/api/winuser/nc-winuser-sendasyncproc)**. Она используется с функцией **[SendMessageCallback](https://learn.microsoft.com/en-us/windows/desktop/api/winuser/nf-winuser-sendmessagecallbacka)**. Система передаёт сообщение функции обратного вызова после передачи сообщения процедуре окна назначения. Тип **SENDASYNCPROC** является указателем на данную функцию обратного вызова. **SendAsyncProc** - прототип имени функции, определяемый приложением.

Возможно отправлять сообщение всем главным (верхний уровень) окнам системы. Например, если приложение изменяет время системы, оно должно сообщить об этом всем главным окнам с помощью сообщения **[WM\_TIMECHANGE](https://learn.microsoft.com/en-us/windows/win32/sysinfo/wm-timechange)**. Приложение может отправить сообщение всем главным окнам, вызвав **[SendMessage](https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-sendmessage)** с указанием **HWND\_TOPMOST** в качестве параметра hwnd. Также можно передавать сообщение всем приложениям с помощью функции **[BroadcastSystemMessage](https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-broadcastsystemmessage)**, указав **BSM\_APPLICATIONS** в параметре **lpdwRecipients**.

Использование функции **[InSendMessage](https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-insendmessage)** или **[InSendMessageEx](https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-insendmessageex)** помогает процедуре окна отпределить, что оно обрабатывает сообщение, отправленное другим потоком. Это необходимо, когда обработка сообщения зависит от происхождения сообщения.

Message Deadlocks (Взаимоблокировки сообщений)

Поток, вызывающий функцию **SendMessage** для отправки сообщения другому потоку, не может продолжить работу до тех пор, пока не вернется процедура окна, принимающая сообщение.

Если принимающий поток передаёт управление во время обработки сообщения, отправляющий поток не может продолжать работу, потому что он ждёт возвращения **SendMessage.** Если принимающий поток прикреплён к той же очереди, что и отправитель, это может привести к взаимоблокировке сообщений. (Обратите внимание, что перехватчики журнала присоединяют потоки к одной и той же очереди.)

Принимающий поток не должен передавать управление явно.

Вызов любой из следующих функций может привести к тому, что поток неявно передаст управление:

**[DialogBox](https://learn.microsoft.com/en-us/windows/desktop/api/winuser/nf-winuser-dialogboxa)**

**[DialogBoxIndirect](https://learn.microsoft.com/en-us/windows/desktop/api/winuser/nf-winuser-dialogboxindirecta)**

**[DialogBoxIndirectParam](https://learn.microsoft.com/en-us/windows/desktop/api/winuser/nf-winuser-dialogboxindirectparama)**

**[DialogBoxParam](https://learn.microsoft.com/en-us/windows/desktop/api/winuser/nf-winuser-dialogboxparama)**

**[GetMessage](https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-getmessage)**

**[MessageBox](https://learn.microsoft.com/en-us/windows/desktop/api/winuser/nf-winuser-messagebox)**

**[PeekMessage](https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-peekmessagea)**

**[SendMessage](https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-sendmessage)**

Для избежания потенциальной взаимоблокировки в приложении рекомендуется использовать функцию **[SendNotifyMessage](https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-sendnotifymessagea)** или **[SendMessageTimeout](https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-sendmessagetimeouta)**. В противном случае процедура окна может определить, было ли полученное ею сообщение отправлено другим потоком, вызвав функцию **[InSendMessage](https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-insendmessage)** или **[InSendMessageEx](https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-insendmessageex)**. До вызова любой из этих функций при обработке сообщения процедура окна должна сначала вызвать **InSendMessage** или **InSendMessageEx**. Если данная функция возвращает **TRUE**, процедура окна должна вызвать функцию **[ReplyMessage](https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-replymessage) до любой другой функции, которая может привести к явному управлению.**

Broadcasting Messages (широковещательная рассылка)

Каждое сообщение состоит из идентификатора сообщения и двух параметров: wParam and lParam. Идентификатор сообщения - уникальное значение, которое определяет цель сообщения. Параметры предоставляют дополнительную информацию о сообщении, а параметр wParam обычно содержит самую большую информацию о сообщении.

Широковещательная рассылка это просто отправка сообщения нескольким получателям в системе. Для создания широковещательной рассылки от приложения нужно использовать функцию **[BroadcastSystemMessage](https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-broadcastsystemmessage)**, определив получателей сообщения. Вместо указания отдельных получателей необходимо указать один или несколько типов получателей: приложения, устанавливаемые драйверы, сетевые драйверы и драйверы устройств системного уровня. Система отправит широковещательную рассылку всем членам каждого указанного типа.

Система обычно передает сообщения в ответ на изменения, которые происходят в драйверах устройств системного уровня или связанных с ними компонентах. Драйвер или связанный с ним компонент передает сообщение приложениям и другим компонентам, чтобы уведомить их об изменении. Например, компонент, отвечающий за дисководы, передает сообщение всякий раз, когда драйвер устройства для дисковода гибких дисков обнаруживает смену носителя, например, когда пользователь вставляет диск в дисковод.

Система рассылает сообщения получателям в следующем порядке: драйверы устройств системного уровня, сетевые драйверы, устанавливаемые драйверы и приложения. Это означает, что драйверы устройств системного уровня, выбранные в качестве получателей, всегда получают возможность первыми ответить на сообщение. В рамках данного типа получателя ни один драйвер не может гарантированно получить данное сообщение раньше любого другого драйвера. Это означает, что сообщение, предназначенное для конкретного драйвера, должно иметь глобальный идентификатор сообщения, чтобы ни один другой драйвер не смог непреднамеренно обработать его.

Возможно также выполнять широковещательную рассылку всем главным окнам, указав HWND\_BROADCAST в функции [SendMessage](https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-sendmessage), [SendMessageCallback](https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-sendmessagecallbacka), [SendMessageTimeout](https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-sendmessagetimeouta), или [SendNotifyMessage](https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-sendnotifymessagea).

Приложения получают сообщения через процедуру окна главных окон. Сообщения не отправляются дочерним окнам. Служебные окна могут получать сообщения через процедуру окна или обработчика служебного окна.

Примечание:

Драйверы устройств системного уровня используют соответствующую системную функцию для трансляции системных сообщений.

Query Messages (сообщения запроса)

Вы можете создавать свои собственные пользовательские сообщения и использовать их для координации действий между вашими приложениями и другими компонентами системы.

Это необходимо, если вы создали собственный драйверы или драйверы устройств системного уровня. Ваши пользовательские сообщения могут передавать информацию вашему драйверу и приложениям, использующим этот драйвер, а также от них.

Чтобы запросить у получателей разрешение на выполнение определенного действия, используйте сообщение запроса. Вы можете создать собственное сообщение запроса, указав значение BSF\_QUERY в параметре dwFlags, вызвав [BroadcastSystemMessage](https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-broadcastsystemmessage). Каждый получатель сообщения запроса должен вернуть TRUE в функцию, чтобы отправить сообщение следующему получателю. Если один получатель отправит BROADCAST\_QUERY\_DENY, передача сообщений прекращается, а функция возвращает ноль.