**Московский авиационный институт**

(национальный исследовательский университет)

**Факультет № 8 «Информационные технологии и прикладная математика»**

**Кафедра 806 «Вычислительная математика и программирование»**

РЕФЕРАТ

по циклу дисциплин «Фундаментальная информатика»

тема работы «Сетевые технологии: Что происходит, при вводе символов в адресную строку браузера?»

Студент: Пермяков Н.А.

Год приема: 2019

Группа: М8О - 108Б - 19

Руководитель: Поповкин А.В.

\_\_.\_\_.\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Что происходит, при вводе символов в адресную строку браузера?

**1. Нажата первая клавиша.**

В этот момент замыкается контур, отвечающий за эту кнопку и ток проходит по логическим контурам клавиатуры. Клавиатура отбрасывает паразитические импульсы и определяет единственно правильный код нажатой клавиши. Например, код клавиши «Enter» - 13. Контроллер кодирует код для передачи в компьютер. Теперь это почти всегда делается через USB или Bluetooth, а раньше в процессе участвовали PS/2 или ADB.

Забегая вперед можно упомянуть, что браузеры уже в это время выполняют алгоритмы автоподстановки, такие как «Predictive Writer» и «Т9».

Некоторые браузеры (например, Rockmelt) предлагают профили друзей на Facebook и в других соц. сетях.

**2. Клавиша «Enter» нажата до конца**

*Если у вас USB-клавиатура:*

* USB-контуру требуется 5 вольт питания, которые поступают через USB-контроллер из компьютера.
* Сгенерированный код клавиши хранится в регистре внутренней памяти клавиатуры, который называется «конечной точкой» (endpoint).
* USB-контроллер компьютера опрашивает эту конечную точку каждые 10 микросекунд и получает хранящийся там код клавиши.
* Затем это значение поступает в USB SIE (Serial Interface Engine) для конвертации в один или более USB-пакетов, которые формируются по низкоуровневому протоколу USB.
* Эти пакеты затем пересылаются с помощью различных электрических сигналов через D+ и D- контакты с максимальной скоростью 1,5 Мб/сек — поскольку HID-устройства (Human Interface Device) всегда были «низкоскоростными».
* Этот последовательный сигнал декодируется в USB-контроллером компьютера и интерпретируется универсальным драйвером HID-устройства (клавиатуры). Затем значение кода клавиши передаётся на «железный» уровень абстракции операционной системы.

**2.1 (На Windows) Сообщение WM\_KEYDOWN отправлено приложению**

HID передаёт событие нажатой клавиши драйверу KBDHID.sys, который конвертирует его в [скан-код](https://ru.wikipedia.org/wiki/%D0%A1%D0%BA%D0%B0%D0%BD-%D0%BA%D0%BE%D0%B4). Клавиша «Enter» имеет скан-код — VK\_RETURN (0x0D). Далее, этот же драйвер связывается с драйвером KBDCLASS.sys (драйвер классов клавиатуры). Он отвечает за безопасную обработку всего ввода с клавиатуры. В дальнейшем этот драйвер вызывает Win32K.sys (после возможной передачи сообщения через установленные сторонние клавиатурные фильтры). Все это происходит в режиме ядра. Win32K.sys определяет, какое окно активно в данный момент, с помощью функции GetForegroundWindow(). Этот API обеспечивает обработку окна адресной строки в браузере. Затем Windows вызывает SendMessage(hWnd, WM\_KEYDOWN, VK\_RETURN, lParam). lParam — это битовая маска, которая указывает на дальнейшую информацию о нажатии клавиши: счётчик повторов (в этом случае 0), актуальный скан-код (может зависеть от OEM, (это производитель оригинального оборудования, то есть условие распространения и обслуживания) но VK\_RETURN обычно не зависит от этого), информацию о том, были ли нажаты дополнительные клавиши (например, Alt, Shift, Ctrl — в нашем случае не были) и некоторые другие данные.

В API Windows есть функция SendMessage, которая помещает сообщение в очередь для конкретного обработчика окон (hWnd). После этого для обработки всех сообщений очереди вызывается главная функция обработки сообщений (WindowProc), присвоенная обработчику hWnd.

Окно (hWnd), активное в данный момент, представляет из себя контрол обработки и в этом случае у WindowsProc есть обработчик для сообщений WM\_KEYDOWN. Этот код изучает третий параметр, который поступил в SendMessage (wParam) и, поскольку это VK\_RETURN, понимает, что пользователь нажал клавишу ENTER.

**2.2 (В GNU/Linux) Сервер Xorg слушает клавиатурные коды**

При использовании графического X server для получения кода будет задействован драйвер evdev. Который преобразует код клавиши в сканкод. После этого символ передаётся в менеджер окон (DWM, metacity, i3, и т.д.), который в свою очередь передаёт символ окну, находящемуся в фокусе. Графический API окна получает символ и выводит соответствующий символ в том окне, в котором находится фокус.

**3. Парсинг URL**

Браузер вытащит из input'а строку с запросом и посмотрит, похоже ли это на адрес. Если да, то добавит недостающие уточнения (например, http, file протокол или порт). Если нет - то скорее всего создаст запрос в поисковую систему, установленную по умолчанию.

**3.1 Список проверки HSTS**

* Браузер проверяет список HSTS (HTTP Strict Transport Security). Это список сайтов, к которым нужно обращаться только по HTTPS. Если сайт в списке, то браузер отправляет запрос через HTTPS.
* В противном случае, начальный запрос посылается по HTTP. (При этом сайт может использовать политику HSTS, но не находиться в списке HSTS — в таком случае на первый запрос по HTTP будет отправлен ответ о том, что необходимо отправлять запросы по HTTPS. Однако это может сделать пользователя уязвимым к [downgrade-атакам](https://en.wikipedia.org/wiki/Moxie_Marlinspike#Notable_research) (понижения версии протокола) чтобы этого избежать, в браузеры и включают список HSTS).

**3.2 Конвертация не-ASCII Unicode символов в название хоста**

* Браузер проверяет имя хоста на наличие символов, отличных от a-z, A-Z, 0-9, -, или ..
* В случае доменного имени google.com никаких проблем не будет, но если бы домен содержал не-ASCII символы, то браузер бы применил кодировку [Punycode](https://en.wikipedia.org/wiki/Punycode) для этой части URL.

Чтобы сделать запрос по указанному URL, браузеру нужно знать IP сервера. Первым делом он смотрим в свой локальный кэш DNS.

*В браузере Chrome локальный кэш DNS доступен по ссылке chrome://net-internals/#dns*

**4. Определение DNS (система именования компьютеров в сети)**

* Браузер проверяет наличие домена в своём кэше.
* Если домена там нет, то браузер вызывает библиотечную функцию gethostbyname для поиска нужного адреса.
* gethostbyname пытается найти нужный адрес в файле hosts.
* Если домен нигде не закэширован и отсутствует в файле hosts, gethostbyname отправляет запрос к сетевому DNS-серверу. В качестве DNS может выступать локальный роутер или DNS-сервер интернет-провайдера у которого тоже есть свой кэш DNS на своих рекурсивных серверах DNS.
* В случае отсутствия записи в кэше на серверах DNS провайдера, запрос идёт на корневой DNS. У корневого DNS тоже есть кэш.
* Если соответствующей записи в кэше корневого DNS нет, запрос идёт дальше по цепочке серверов DNS.
* Если DNS-сервер находится в той же подсети, то [ARP-запрос](https://ru.wikipedia.org/wiki/ARP) (Address Resolution Protocol - протокол определения адреса) отправляется этому серверу.
* Если DNS-сервер находится в другой подсети, то ARP-запрос отправляется на IP-адрес шлюза по умолчанию.

К примеру, если адрес нашего сайта site.com.ua, то запросы к DNS выглядят так: **site.com.ua.**

(корневой DNS) -> **ua**

(DNS зоны “ua”) -> **com**

(DNS зоны “com”)-> **site.**

Если на любом из этапов находится нужная запись, то она сохраняется во всех кэшах и управление возвращается браузеру, который уже знает IP нужного сервера.

**4.1 Процесс отправки ARP-запроса**

Для отправки широковещательного запроса ARP, сетевому стеку нужно узнать IP-адрес получателя и MAC-адрес интерфейса, который будет для этого использован.

Сначала проверяется кэш ARP на предмет наличия IP получателя. Если он есть в кэше, возвращается результат «IP получателя <=> MAC».

Если записи в кэше нет:

* Проверяется таблица маршрутизации — это делается для того, чтобы узнать, есть ли искомый IP-адрес в какой-либо из подсетей локальной таблицы. Если он там, то запрос посылается с помощью интерфейса, связанного с этой подсетью.
* Если адрес в таблице не обнаружен - используется интерфейс подсети основного шлюза.
* Определяется MAC-адрес выбранного сетевого интерфейса.
* Отправляется ARP-запрос (второй уровень стека):

ARP-запрос:

Sender MAC: interface:mac:address:here

Sender IP: interface.ip.goes.here

Target MAC: FF:FF:FF:FF:FF:FF (Broadcast)

Target IP: target.ip.goes.here

В зависимости от того, какое «железо» расположено между компьютером и роутером (маршрутизатором):

* Если компьютер напрямую подключён к роутеру, то это устройство отправляет ARP-ответ (ARP Reply).
* Если компьютер подключён к хабу, (повторитель, на который даётся один IP адрес), то отправляет широковещательный ARP-запрос со всех своих портов.
* Если роутер подключён по тому же «проводу», то отправит ARP-ответ.
* Если компьютер соединён со свитчом (коммутатор, имеющий несколько отдельных IP адресов. Этим самым снижается нагрузка на сеть и каждый компьютер получит лишь то, что ему нужно), то он проверит локальную CAM/MAC-таблицу, чтобы узнать, какой порт в ней имеет нужный MAC-адрес. Если нужного адреса в таблице нет, то он заново отправит широковещательный ARP-запрос по всем портам.
* Если в таблице есть нужная запись, то свитч отправит ARP-запрос на порт с искомым MAC-адресом.
* Если роутер «на одной линии» со свитчем, то он ответит (ARP Reply).

Теперь у сетевой библиотеки есть IP-адрес либо DNS-сервера либо основного шлюза. Продолжается процесс распознавания домена.

* Порт 53 открывается для отправки UDP-запроса к DNS-серверу (если размер ответа велик, будет использован TCP - Transmission Control Protocol основной протокол передачи данных по сети Интернет).
* Если информации у DNS-сервера не оказывается, то запрашивается рекурсивный поиск, который проходит по списку DNS-серверов, пока не доходит до SOA-записи (Start of Authority - начальная запись зоны, которая указывает местоположение эталонной записи о домене) и не находится нужный ответ.

**5. Открытие сокета**

После того как браузер получает IP-адрес конечного сервера и данные о порте из URL (80 порт для HTTP, 443 для HTTPS) он перенаправляет их как параметры для вызова функции *socket* и запрашивает поток TCP *socket stream* — *AF\_INET* и *SOCK\_STREAM*.

* Этот запрос сначала проходит через транспортный уровень, где собирается TCP-сегмент. В заголовок добавляется порт назначения, исходный порт выбирается из динамического списка портов ядра.
* Получившийся сегмент отправляется на сетевой уровень, где ему добавляют IP-заголовок, в котором содержится ip-адрес сервера назначения и ip-адрес нашего компьютера.
* Пакет передаётся на канальный уровень. Добавляется заголовок кадра, включающий MAC-адрес сетевой карты компьютера, а также MAC-адрес шлюза (локального роутера).
* Если ядру ничего не известно о MAC-адресе шлюза, то для его нахождения отправляется широковещательный ARP-запрос.

Пакет готов к передаче по [Ethernet](http://en.wikipedia.org/wiki/IEEE_802.3), [WiFi и](https://en.wikipedia.org/wiki/IEEE_802.11)[сотовой связи](https://en.wikipedia.org/wiki/Cellular_data_communication_protocol).

В большинстве случаев пакет из компьютера проходит по локальной сети, затем попадает в модем (преобразует сигнал для его «физического» распространения), где превращается из цифрового в аналоговый сигнал. На другой стороне соединения расположен другой модем, который конвертирует аналоговый сигнал в цифровые данные и передаёт их следующему [сетевому узлу](https://en.wikipedia.org/wiki/Computer_network#Network_nodes), где происходит дальнейший анализ данных об отправителе и получателе.

Иногда пакет отправляется сразу через Ethernet или оптику, тогда он остаётся цифровым и доходит до следующего узла сети. В конце концов сигнал доходит до роутера локальной подсети. Затем от одного роутера к другому, пока не доберётся до сервера назначения. Каждый маршрутизатор на пути будет извлекать адрес назначения из IP-заголовка и отправлять пакет на следующий хоп. Значение поля TTL (Time To Live) в IP-заголовке будет каждый раз уменьшаться после прохождения каждого роутера. Если значение поля TTL достигнет нуля, пакет будет отброшен.

Отправка и получение пакетов происходят многократно в рамках соединения по TCP.

**5.1 Жизненный цикл TCP-соединения**

a. Клиент выбирает номер начальной последовательности (ISN) и отправляет пакет серверу с установленным битом SYN для открытия соединения.

b. Сервер получает пакет с битом SYN и, если готов к установлению соединения, то:

* Выбирает собственный номер начальной последовательности;
* Устанавливает SYN-бит, чтобы сообщить о выборе начальной последовательности;
* Копирует ISN + 1 в поле номера подтверждения (ACK) и добавляет ACK-флаг для обозначения подтверждения получения первого пакета.

c. Клиент подтверждает соединение путём отправки пакета:

* Увеличивает номер своей начальной последовательности;
* Увеличивает номер подтверждения получения;
* Устанавливает поле ACK.

d. Данные передаются следующим образом:

* Когда одна сторона отправляет N байтов, то увеличивает значение поля SEQ (порядковое значение) на это число.
* Когда вторая сторона подтверждает получение этого пакета (или цепочки пакетов), она отправляет пакет ACK, в котором значение поля ACK равняется последней полученной последовательности.

e. Закрытие соединения:

* Сторона, которая хочет закрыть соединение, отправляет пакет FIN (он последний);
* Другая сторона подтверждает FIN (с помощью ACK) и отправляет собственный FIN-пакет;
* Инициатор прекращения соединения подтверждает получение FIN отправкой собственного ACK.

**6. TLS рукопожатие (handshake)**

* Клиентский компьютер отправляет сообщение ClientHello серверу со своей версией протокола [TLS](https://ru.wikipedia.org/wiki/TLS) ([Transport Layer Security](https://en.wikipedia.org/wiki/Transport_Layer_Security) - криптографический протокол), списком поддерживаемых алгоритмов шифрования и методов компрессии данных.
* Сервер отвечает клиенту сообщением ServerHello, содержащим версию TLS, выбранный метод шифрования, выбранные методы компрессии и публичный сертификат сервиса, подписанный центром сертификации (CA). Сертификат содержит публичный ключ, который будет использоваться клиентом для шифрования оставшейся части процедуры «рукопожатия», пока не будет согласован симметричный ключ.
* Если проверка проходит, клиент создаёт последовательность псевдослучайных байтов и шифрует её с публичным ключом сервера. Эта последовательность используется для создания симметричного ключа.
* Сервер расшифровывает случайные байты с помощью своего секретного ключа и использует эти байты для генерации своей копии симметричного ключа.
* Клиент отправляет серверу сообщение Finished, шифруя хеш передачи с помощью симметричного ключа.
* Сервер генерирует собственный хеш, а затем расшифровывает полученный от клиента хеш, чтобы проверить, совпадёт ли он с собственным. Если совпадение обнаружено, сервер отправляет клиенту собственный ответ Finished, также зашифрованный симметричным ключом.
* После этого TLS-сессия передаёт данные приложения (HTTP), зашифрованные с помощью подтверждённого симметричного ключа.

**7. Протокол HTTP**

* Если используемый браузер был создан Google, то вместо отправки HTTP-запроса для получения страницы, он отправит запрос, чтобы попытаться «договориться» с сервером об «апгрейде» протокола с HTTP до [SPDY](https://ru.wikipedia.org/wiki/SPDY) («спиди» - протокол, разработанный Google).
* Если клиент использует HTTP-протокол и не поддерживает SPDY, то отправляет серверу запрос следующей формы:

GET / HTTP/1.1

Host: google.com

Connection: close

[другие заголовки]

где [другие заголовки] — это серия пар «ключ: значение», разбитых переносом строки. Если клиент использует более старые протоколы, чем HTTP/1.1, то в запросе не будет заголовка Host и версия в первой строчке будет другая – 1.0 или 0.9.

В протоколе HTTP/1.1 задана возможность закрывать соединение после окончания получения ответа. Заголовок выглядит так:

Connection: close

Если клиент не поддерживает постоянные соединения, он обязан включать такой заголовок в любой запрос. После отправки запроса и заголовков, браузер отправляет серверу единичную пустую строку, сигнализируя о том, что содержимое сообщения закончилось.

Сервер отвечает специальным кодом, который обозначает статус запроса:

200 OK

[заголовки ответа]

В конце посылается пустая строка, а затем оставшийся контент HTML-страницы [www.google.com](http://www.google.com/). Сервер может затем закрыть соединение, или, оставить открытым, если так запрашивает клиент.

Если HTTP-заголовки отправленные веб-браузером включают информацию, которой серверу достаточно для определения версии файла, закэшированного в браузере и этот файл не менялся со времени последнего запроса, то ответ может принять следующую форму:

304 Not Modified

[заголовки ответа]

и, соответственно, если клиенту не посылается никакого контента, тогда браузер загружает HTML-файл из кэша.

После разбора HTML, браузер (и сервер) повторяет процесс загрузки для каждого ресурса (изображения, стили, скрипты, favicon.ico и так далее), на который ссылается HTML-страница, но при этом изменяется адрес каждого запроса c *GET / HTTP/1.1* на *GET /$(относительный URL ресурса www.google.com) HTTP/1.1.* Если HTML ссылается на ресурс, размещённый на домене, отличном от google.com, то браузер возвращается к шагам, включающим разрешение доменного имени, а затем заново проходит процесс до текущего состояния, но уже для другого домена. Заголовок Host в запросе вместо google.com будет установлен на нужное доменное имя.

**7.1 Обработка HTTP-запросов на сервере**

HTTPD (HTTP Daemon) является одним из инструментов обработки запросов/ответов на стороне сервера. Наиболее популярные HTTPD-серверы это *Apache* или *Nginx* для Linux и *IIS* для Windows.

— HTTPD (HTTP Daemon) получает запрос.

— Сервер разбирает запрос по следующим параметрам:

* Метод HTTP-запроса (GET, POST, HEAD, PUT или DELETE). В случае URL-адреса, который пользователь напечатал в строке браузера, мы имеем дело с GET-запросом.
* Домен. В нашем случае — google.com.
* Запрашиваемый путь/страницы, в нашем случае — / (нет запрошенного пути, / — это путь по умолчанию).

— Сервер проверяет существование виртуального хоста, который соответствует google.com.

— Сервер проверяет, что google.com может принимать GET-запросы.

— Сервер проверяет, имеет ли клиент право использовать этот метод (на основе IP-адреса, аутентификации и прочее).

— Если на сервере установлен модуль перезаписи, то он сопоставляет запрос с одним из сконфигурированных правил. Если находится совпадающее правило, то сервер использует его, чтобы переписать запрос.

— Сервер находит контент, который соответствует запросу, в нашем случае он изучит индексный файл.

— Далее сервер разбирает («парсит») файл с помощью обработчика. Если Google работает на PHP, то сервер использует PHP для интерпретации индексного файла и направляет результат клиенту.

**8. За кулисами браузера**

После получения браузером всех ресурсов (HTML, CSS, JS, картинки, и т.д.) браузер парсит текстовые ресурсы (HTML, CSS, JS), и запускает рендер страницы. Для этого строится дерево DOM (Document Object Model), затем оно обрабатывается, просчитывается расположение элементов и они выводятся на экран.

Функция браузера – предоставить выбранный веб-ресурс, запросив его с сервера, и показать его в окне. Обычно это HTML-документ, но это может быть и PDF, картинка и другой контент. Местоположение ресурса задаётся универсальным идентификатором URI.

Способ, который браузер использует для интерпретации и отображения HTML-файлов описан в спецификациях HTML и CSS. Эти документы разработаны и поддерживаются консорциумом W3C (World Wide Wib Consortium), которая занимается стандартизацией веба.

Почти все браузеры содержат:

* Адресная строка, куда вставляются URL-адреса;
* Кнопки возврата на предыдущую и следующую страницу;
* Возможность создания закладок;
* Кнопки обновления страницы (рефреш) и остановки загрузки текущих документов;
* Кнопка «домой», возвращающая пользователя на домашнюю страницу.

**Высокоуровневая структура браузера**

* **Пользовательский интерфейс**: В него входит адресная строка, кнопки продвижения вперёд/назад, меню закладок и т. д..
* **«Движок» браузера**: Распределяет действия между движком рендеринга и интерфейсом пользователя.
* **«Движок» рендеринга**: Отвечает за отображение запрашиваемого контента. К примеру, если запрашивается HTML, то «движок» разбирает код HTML и CSS, а затем отображает полученный контент на экране.
* **Сетевая часть**: с помощью сетевых функций браузер обрабатывает вызовы, вроде HTTP-запросов, с применением различных реализаций для разных платформ.
* **Бэкенд интерфейса (UI)**: Используется для отрисовки базовых виджетов, вроде комбо-боксов и окон.
* **Интерпретатор JavaScript**: Используется для парсинга и выполнения JavaScript-кода.
* **Хранилище данных**: Браузеру может понадобиться локально хранить некоторые данные (например, cookie). Кроме того, браузеры поддерживают различные механизмы хранения, такие как localStorage, IndexedDB, WebSQL и FileSystem.

**9. Парсинг HTML**

Движок рендеринга начинает получать содержимое запрашиваемого документа от сетевого механизма браузера кусками по 8Кб. Главной задачей HTML-парсера является разбор разметки в специальное дерево. Получающееся на выходе дерево («parse tree») - это дерево DOM-элементов и узлов атрибутов. Это модель объектного представления HTML-документа и интерфейс для взаимодействия HTML-элементов с беком (например, JavaScript-кодом). До обработки скриптов DOM имеет почти полное взаимно однозначное соответствие с разметкой.

**Алгоритм разбора**

HTML-нельзя «распарсить» с помощью обычных анализаторов (нисходящих или восходящих). Разметка слишком либеральная, браузеры допускают определённые ошибки в разметке, а работа скриптов иногда принуждает процесс разбора возвращаться к уже пройденным местам. В связи с этим процесс разбора иногда меняет входные данные. Подробнее алгоритм разбора описан в спецификации HTML5. Алгоритм состоит из двух этапов: токенизации и создания дерева.

**Действия после завершения парсинга**

После этого браузер начинает подгружать внешние ресурсы, связанные со страницей (стили, изображения, скрипты и так далее). На этом этапе браузер помечает документ, как интерактивный и начинает разбирать скрипты, находящиеся в «отложенном» состоянии: то есть те из них, что должны быть исполнены после парсинга. После этого статус документа устанавливается в состояние «complete» и инициируется событие загрузки («load»).

Примечание: ошибки «Invalid Syntax» при разборе не может быть, поскольку браузеры исправляют любой «невалидный» контент и продолжают работу.

**10. Интерпретация CSS**

* Во время разбора браузер парсит CSS-файлы, содержимое тегов <style> и атрибутов «style» c помощью «[лексической и синтаксической грамматики CSS](http://www.w3.org/TR/CSS2/grammar.html)».
* Каждый CSS-файл разбирается в объект StyleSheet, каждый из таких объектов содержит правила CSS с селекторами и объектами в соответствии с грамматикой CSS.
* Парсер CSS может быть как восходящим, так и нисходящим.

**11. Рендеринг страниц**

* Путём перебора DOM-узлов и вычисления для каждого узла значений CSS-стилей создаётся «Дерево рендера» (Render Tree или Frame Tree).
* Вычисляется предпочтительная ширина каждого узла в нижней части дерева — для этого суммируются значения предпочтительной ширины дочерних узлов, а также горизонтальные поля, границы и отступы узлов.
* Вычисляется реальная ширина каждого узла сверху-вниз (доступная ширина каждого узла выделяется его потомкам).
* Вычисляется высота каждого узла снизу-вверх — для этого применяется перенос текста и суммируются значения полей, высоты, отступов и границ потомков.
* Вычисляются координаты каждого узла (с использованием ранее полученной информации).
* Если элементы плавающие или спозиционированы абсолютно или относительно, предпринимаются более сложные действия.
* Создаются слои для описания того, какие части страницы можно анимировать без необходимости повторного растрирования. Каждый объект (фрейма или рендера) присваивается слою.
* Для каждого слоя на странице выделяются текстуры.
* Объекты (рендеры/фреймы) каждого слоя перебираются и для соответствующих слоёв выполняются команды отрисовки. Растрирование может осуществляться процессором или возможна отрисовка на графическом процессоре (GPU) через D2D/SkiaGL.
* Все вышеперечисленные шаги могут требовать повторного использования значений, сохранённых с последнего рендеринга страницы, такая инкрементальная работа требует меньше затрат.
* Слои страницы отправляются процессу-компоновщику, где они комбинируются со слоями для другого видимого контента (интерфейс браузера, iframe-элементы, addon-панели).
* Вычисляются финальные позиции слоёв и через Direct3D/OpenGL отдаются композитные команды. Командные буферы GPU освобождаются для асинхронного рендеринга и фрейм отправляется для отображения на экран.

**12. Рендеринг GPU**

* Во время процесса рендеринга уровни графических вычислений могут использовать процессор компьютера или графический процессор (GPU).
* Во втором случае уровни графического программного обеспечения делят задачу на множество частей, что позволяет использовать параллелизм GPU для вычисления плавающей точки, которое требуется для процесса рендеринга.

**13. Вызванное пользователем и пост-рендеринговое исполнение**

После завершения рендеринга, браузер исполняет JavaScript-код в результате срабатывания некоего часового механизма (так работают дудлы на странице Google) или в результате действий пользователя (ввод поискового запроса в строку и получение рекомендаций в ответ). Также могут срабатывать плагины вроде Flash или Java (но не в рассматриваемом примере с домашней страницей Google). Скрипты могут потребовать обработки дополнительных сетевых запросов, изменять страницу или её шаблон, что приведёт к следующему этапу рендеринга и отрисовки.