Webserv

Именно тогда вы наконец поймете, почему URL начинается с HTTP

Резюме:
Этот проект посвящен написанию собственного HTTP-сервера. Вы сможете протестировать его с помощью реального браузера.
HTTP - один из самых используемых протоколов в Интернете. Знание его тонкостей будет полезно, даже если вы не будете работать над веб-сайтом.

Версия: 20.2

Содержание

Глава I

Введение

Протокол передачи гипертекста (HTTP) - это прикладной протокол для распределенных, совместных, гипермедийных информационных систем.

HTTP - это основа передачи данных во Всемирной паутине, где гипертекстовые документы содержат гиперссылки на другие ресурсы, к которым пользователь может легко получить доступ. Например, щелчком мыши или касанием экрана в веб-браузере.

HTTP был разработан для создания гипертекста и Всемирной паутины.

Основной функцией веб-сервера является хранение, обработка и доставка веб-страниц клиентам. Связь между клиентом и сервером осуществляется с помощью протокола передачи гипертекста (HTTP).

Доставляемые страницы чаще всего представляют собой HTML-документы, которые могут включать в себя изображения, таблицы стилей и скрипты в дополнение к текстовому содержанию.

Для сайта с высокой посещаемостью может использоваться несколько веб-серверов.

Агент пользователя, обычно веб-браузер или веб-гусеница, инициирует взаимодействие, запрашивая определенный ресурс с помощью HTTP, а сервер отвечает содержимым этого ресурса или сообщением об ошибке, если не может этого сделать. Ресурс обычно представляет собой реальный файл на вторичном хранилище сервера, но это не обязательно так и зависит от того, как реализован веб-сервер.

Хотя основной функцией является передача содержимого, полная реализация HTTP также включает способы получения содержимого от клиентов. Эта функция используется для отправки веб-форм, включая загрузку файлов.

Глава II Общие правила

- Ваша программа не должна аварийно завершаться ни при каких обстоятельствах (даже когда у нее закончится память), и не должна неожиданно завершаться.
  Если это произойдет, ваш проект будет считаться нефункциональным, и ваша оценка будет
  0.
- Вы должны создать Makefile, который будет компилировать ваши исходные файлы. Он не должен перелинковываться.
- Ваш Makefile должен, по крайней мере, содержать эти правила:
  $(NAME), all, clean, fclean и re.
- Скомпилируйте ваш код с помощью c++ и флагов -Wall -Wextra -Werror
- Ваш код должен соответствовать **стандарту C++ 98**. Тогда он должен компилироваться, если вы добавите флаг -std=c++98.
- Старайтесь всегда разрабатывать с использованием наиболее доступных функций C++ (например, выбирайте
  <cstring> над <string.h>). Вы можете использовать функции языка C, но всегда предпочитайте их версии на C++, если это возможно.
- Любые внешние библиотеки и библиотеки Boost запрещены.

Глава III

Обязательная часть

Название программы	webserv
Сдать файлы	Makefile, *.{h, hpp}, *.cpp, *.tpp, *.ipp, конфигурационные файлы
Makefile	NAME, all, clean, fclean, re
Аргументы	[Конфигурационный файл]
Внешние функции.	Все на C++ 98. execve, dup, dup2, pipe, strerror, gai_strerror, errno, dup, dup2, fork, htons, htonl, ntohs, ntohl, select, poll, epoll (epoll_create, epoll_ctl, epoll_wait), kqueue (kqueue, kevent), socket, accept, listen, send, recv, bind, connect, getaddrinfo, freeaddrinfo, setsockopt, getsockname, getprotobyname, fcntl
Либфт уполномочен	н/а
Описание	HTTP-сервер на C++ 98

Вы должны написать HTTP-сервер на C++ 98.

Ваш исполняемый файл будет запущен следующим образом:

./webserv [файл конфигурации]

Даже если poll() упоминается в теме и шкале оценки, вы можете использовать любой эквивалент, такой как select(), kqueue() или epoll().

Пожалуйста, прочитайте RFC и проведите несколько тестов с помощью telnet и NGINX перед началом этого проекта.
Даже если вам не нужно реализовывать все RFC, их прочтение поможет вам разработать необходимые функции.

III.1 Требования

- Ваша программа должна принимать конфигурационный файл в качестве аргумента или использовать путь по умолчанию.
- Вы не можете исключить другой веб-сервер.
- Ваш сервер никогда не должен блокироваться, и при необходимости клиент может быть отшит должным образом.
- Он должен быть неблокирующим и использовать только **1** poll() (или эквивалент) для всех операций ввода-вывода между клиентом и сервером (включая прослушивание).
- poll() (или эквивалент) должен проверять чтение и запись одновременно.
- Вы никогда не должны выполнять операции чтения или записи, не пройдя через poll() (или эквивалент).
- Проверять значение errno после операции чтения или записи строго запрещено.
- Вам не нужно использовать poll() (или эквивалент) перед чтением файла конфигурации.

Поскольку вы должны использовать неблокирующие дескрипторы файлов, можно использовать функции read/recv или write/send без poll() (или эквивалента), и ваш сервер не будет блокироваться.
Но это потребует больше системных ресурсов.
Таким образом, если вы попытаетесь прочитать/отправить или записать/отправить в любом файловом дескрипторе без использования poll() (или эквивалента), ваша оценка будет равна 0.

- Вы можете использовать все макросы и определения, такие как FD_SET, FD_CLR, FD_ISSET, FD_ZERO (понимание того, что и как они делают, очень полезно).
- Запрос к вашему серверу никогда не должен зависать надолго.
- Ваш сервер должен быть совместим с выбранным вами **веб-браузером.**
- Мы рассмотрим, что NGINX совместим с HTTP 1.1 и может быть использован для сравнения заголовков и поведения ответов.
- Ваши коды состояния HTTP-ответов должны быть точными.
- На вашем сервере должны быть **страницы ошибок по умолчанию,** если таковые не предусмотрены.
- Вы не можете использовать fork для чего-то другого, кроме CGI (например, PHP, или Python, и так далее).
- Вы должны уметь **обслуживать полностью статичный сайт**.
- Клиенты должны иметь возможность **загружать файлы.**

- Вам нужны как минимум методы GET, POST и DELETE.
- Стресс-тесты вашего сервера. Он должен оставаться доступным любой ценой.
- Ваш сервер должен иметь возможность прослушивать несколько портов (см. *файл конфигурации*).

III.2 Только для MacOS

Поскольку в MacOS функция write() реализована не так, как в других ОС Unix, вам разрешено использовать fcntl().
Вы должны использовать файловые дескрипторы в неблокирующем режиме, чтобы получить поведение, похожее на поведение других ОС Unix.

Однако вам разрешено использовать fcntl() только следующим образом: fcntl(fd, F_SETFL, O_NONBLOCK);
Любой другой флаг запрещен.

III.3 Конфигурационный файл

Вы можете черпать вдохновение из части 'server' конфигурационного файла NGINX.

В конфигурационном файле вы должны быть в состоянии:

- Выберите порт и хост каждого "сервера".
- Устанавливать имена_серверов или нет.
- Первый сервер для хоста:порта будет по умолчанию для этого хоста:порта (это означает, что он будет отвечать на все запросы, которые не принадлежат другому серверу).
- Настройка страниц ошибок по умолчанию.
- Ограничьте размер тела клиента.
- Установите маршруты с одним или несколькими из следующих правил/конфигураций (маршруты не будут использовать regexp):
    - Определите список принятых методов HTTP для маршрута.
    - Определите перенаправление HTTP.
    - Определите каталог или файл, откуда следует искать файл (например, если url /kapouet укоренен в /tmp/www, url /kapouet/pouic/toto/pouet является /tmp/www/pouic/toto/pouet).
    - Включить или выключить листинг каталога.

- Установите файл по умолчанию для ответа, если запрос является каталогом.
- Выполнение CGI на основе определенного расширения файла (например, .php).
- Сделайте маршрут способным принимать загруженные файлы и настройте место их сохранения.
  * Вам интересно, что такое CGI?
  * Поскольку вы не будете вызывать CGI напрямую, используйте полный путь в качестве PATH_INFO.
  * Просто помните, что для кускового запроса ваш сервер должен распаковать его, а CGI будет ожидать EOF как конец тела.
  * То же самое для вывода CGI. Если из CGI не возвращается content_length, EOF будет отмечать конец возвращаемых данных.
  * Ваша программа должна вызвать CGI с запрошенным файлом в качестве первого аргумента.
  * CGI должен быть запущен в правильном каталоге для доступа к файлам по относительному пути.
  * Ваш сервер должен работать с одним CGI (php-CGI, Python и так далее).

Вы должны предоставить некоторые файлы конфигурации и базовые файлы по умолчанию для тестирования и демонстрации работы каждой функции во время оценки.

Если у вас возник вопрос о каком-то одном поведении, вам следует сравнить поведение вашей программы с поведением NGINX.
Например, проверьте, как работает имя_сервера.
Мы поделились с вами небольшим тестером. Проходить его не обязательно, если все работает нормально с вашим браузером и тестами, но он может помочь вам найти некоторые ошибки.

Главное - это устойчивость. Ваш сервер никогда не должен умереть.

Не тестируйте с помощью только одной программы. Пишите тесты на более удобном языке, таком как Python или Golang и так далее. Даже на C или C++, если хотите.

Глава IV

Бонусная часть

Вот дополнительные функции, которые вы можете добавить:

- Поддержка cookies и управления сессиями (подготовьте краткие примеры).
- Обработка нескольких CGI.

Бонусная часть оценивается только в том случае, если обязательная часть выполнена безупречно. Совершенство означает, что обязательная часть выполнена полностью и работает без сбоев. Если вы не выполнили ВСЕ обязательные требования, ваша бонусная часть не будет оцениваться вообще.

Глава V

Представление и экспертная оценка

Сдайте задание в свой Git-репозиторий, как обычно. Во время защиты будет оцениваться только та работа, которая находится в вашем репозитории. Не стесняйтесь дважды проверять имена файлов, чтобы убедиться в их правильности.