



Servidor Web

É quando você finalmente entende por que uma URL começa com HTTP

Resumo:

Este projeto é sobre escrever seu próprio servidor HTTP.

Você poderá testá-lo com um navegador real.

HTTP é um dos protocolos mais utilizados na internet.

Conhecer seus mistérios será útil, mesmo que você não esteja trabalhando em um site.

Versão: 21.1

Conteúdo

I	Introdução	2
II	Regras gerais	3
III	Parte obrigatória	4
III.1	Requisitos	6
III.2	Somente para MacOS	7
III.3	Arquivo de configuração	7
IV	Parte bônus	9
V	Submissão e avaliação por pares	10

Capítulo I

Introdução

O **Hypertext Transfer Protocol** (HTTP) é um protocolo de aplicação para sistemas de informação hipermídia distribuídos e colaborativos.

HTTP é a base da comunicação de dados para a World Wide Web, onde documentos de hipertexto incluem hiperlinks para outros recursos que o usuário pode acessar facilmente. Por exemplo, com um clique do mouse ou tocando na tela de um navegador da web.

O HTTP foi desenvolvido para facilitar o hipertexto e a World Wide Web.

A função primária de um servidor web é armazenar, processar e entregar páginas web aos clientes. A comunicação entre cliente e servidor ocorre usando o Hypertext Transfer Protocol (HTTP).

As páginas entregues são, na maioria das vezes, documentos HTML, que podem incluir imagens, folhas de estilo e scripts, além do conteúdo do texto.

Vários servidores web podem ser usados para um site de alto tráfego.

Um agente de usuário, geralmente um navegador da web ou rastreador da web, inicia a comunicação solicitando um recurso específico usando HTTP e o servidor responde com o conteúdo desse recurso ou uma mensagem de erro se não for possível fazê-lo. O recurso é normalmente um arquivo real no armazenamento secundário do servidor, mas esse não é necessariamente o caso e depende de como o servidor da web é implementado.

Embora a função primária seja servir conteúdo, a implementação completa do HTTP também inclui maneiras de receber conteúdo de clientes. Esse recurso é usado para enviar formulários da web, incluindo o upload de arquivos.

Capítulo II

Regras gerais

- Seu programa não deve travar em nenhuma circunstância (mesmo quando ficar sem memória) e não deve parar inesperadamente.
Se isso acontecer, seu projeto será considerado não funcional e sua nota será 0.
- Você tem que entregar um Makefile que irá compilar seus arquivos de origem. Ele não deve revincular.
- Seu Makefile deve conter pelo menos as regras:
\$(NAME), todos, limpo, fclean e re.
- Compile seu código com c++ e os sinalizadores -Wall -Wextra -Werror
- Seu código deve estar em conformidade com o **padrão C++ 98**. Então, ele ainda deve compilar se você adicionar o sinalizador -std=c++98.
- Tente sempre desenvolver usando o máximo de recursos C++ que puder (por exemplo, escolha <cstring> em vez de <string.h>). Você tem permissão para usar funções C, mas sempre prefira suas versões C++, se possível.
- Qualquer biblioteca externa e bibliotecas Boost são proibidas.

Capítulo III

Parte obrigatória

Nome do programa	servidor web
Entregar arquivos	Makefile, *.h, *.hpp, *.cpp, *.tpp, *.ipp, arquivos de configuração
Faça o arquivo	NOME, tudo, limpo, fclean, re
Argumentos	[Um arquivo de configuração]
Funções externas.	Tudo em C++ 98. executar, dup, dup2, pipe, strerror, gai_strerror, errno, dup, dup2, garfo, socketpair, htons, htonl, ntohs, ntohl, selecionar, pesquisar, epoll (epoll_create, epoll_ctl, epoll_wait), kqueue (kqueue, kevent), socket, aceitar, ouvir, enviar, receber, chdir bind, conectar, obteraddrinfo, freeaddrinfo, setsockopt, getsockname, getprotobyname, fcntl, fechar, ler, escrever, waitpid, matar, sinalizar, acessar, stat, abrir, open, readdir e closedir.
Autorizado pela Libft	n / D
Descrição	Um servidor HTTP em C++ 98

Você deve escrever um servidor HTTP em C++ 98.

Seu executável será executado da seguinte maneira:

./webserv [arquivo de configuração]



Mesmo que poll() seja mencionado no assunto e na escala de avaliação, você pode usar qualquer equivalente, como select(), kqueue() ou epoll().



Por favor, leia o RFC e faça alguns testes com telnet e NGINX antes iniciando este projeto.

Mesmo que você não precise implementar todo o RFC, lê-lo ajudará você a desenvolver os recursos necessários.

III.1 Requisitos

- Seu programa deve receber um arquivo de configuração como argumento ou usar um caminho padrão.
- Você não pode executar outro servidor web.
- Seu servidor nunca deve bloquear e o cliente pode ser rejeitado corretamente, se necessário.
- Deve ser não bloqueante e usar apenas **1 poll()** (ou equivalente) para todas as operações de E/S entre o cliente e o servidor (incluindo escuta).
- poll() (ou equivalente) deve verificar leitura e escrita ao mesmo tempo.
- Você nunca deve fazer uma operação de leitura ou gravação sem passar por poll() (ou equivalente).
- Verificar o valor de erro é estritamente proibido após uma operação de leitura ou escrita.
- Você não precisa usar poll() (ou equivalente) antes de ler seu arquivo de configuração.



Como você precisa usar descritores de arquivo não bloqueantes, é possível usar funções de leitura/recepção ou gravação/envio sem poll() (ou equivalente), e seu servidor não estaria bloqueando. Mas isso consumiria mais recursos do sistema. Portanto, se você tentar ler/receber ou escrever/enviar qualquer descritor de arquivo sem usar poll() (ou equivalente), sua nota será 0.

- Você pode usar qualquer macro e definir como FD_SET, FD_CLR, FD_ISSET, FD_ZERO (entender o que e como eles fazem isso é muito útil).
- Uma solicitação ao seu servidor nunca deve ficar travada para sempre.
- Seu servidor deve ser compatível com o **navegador** da sua escolha.
- Consideraremos que o NGINX é compatível com HTTP 1.1 e pode ser usado para comparar cabeçalhos e comportamentos de resposta.
- Seus códigos de status de resposta HTTP devem ser precisos.
- Seu servidor deve ter **páginas de erro padrão** caso nenhuma seja fornecida.
- Você não pode usar fork para algo diferente de CGI (como PHP, Python e assim por diante).
- Você deve ser capaz de **servir um site totalmente estático**.
- Os clientes devem poder fazer **upload de arquivos**.
- Você precisa de pelo menos métodos GET, POST e DELETE.
- Teste o estresse do seu servidor. Ele deve permanecer disponível a todo custo.
- Seu servidor deve ser capaz de escutar várias portas (veja o *arquivo de configuração*).

III.2 Somente para MacOS



Como o MacOS não implementa `write()` da mesma forma que outros sistemas operacionais Unix, você tem permissão para usar `fcntl()`.
Você deve usar descritores de arquivo no modo não bloqueante para obter um comportamento semelhante ao de outros sistemas operacionais Unix.



Entretanto, você tem permissão para usar `fcntl()` somente da seguinte maneira: `fcntl(fd, F_SETFL, O_NONBLOCK, FD_CLOEXEC);`
Qualquer outra bandeira é proibida.

III.3 Arquivo de configuração



Você pode obter alguma inspiração na parte "servidor" do arquivo de configuração do NGINX.

No arquivo de configuração, você deve ser capaz de:

- Escolha a porta e o host de cada 'servidor'.
- Configurar os `server_names` ou não.
- O primeiro servidor para um `host:porta` será o padrão para este `host:porta` (isso significa que ele responderá a todas as solicitações que não pertencem a outro servidor).
- Configurar páginas de erro padrão.
- Limite o tamanho do corpo do cliente.
- Configure rotas com uma ou várias das seguintes regras/configurações (as rotas não serão estar usando regexp):
 - Defina uma lista de métodos HTTP aceitos para a rota.
 - Defina um redirecionamento HTTP.
 - Defina um diretório ou um arquivo de onde o arquivo deve ser pesquisado (por exemplo, se url / kapouet estiver enraizado em /tmp/www, url /kapouet/pouic/toto/pouet é /tmp/www/pouic/toto/pouet).
 - Ative ou desative a listagem de diretórios.

- Defina um arquivo padrão para responder se a solicitação for um diretório.
- Execute CGI com base em determinada extensão de arquivo (por exemplo .php).
- Faça funcionar com os métodos POST e GET.
- Tornar a rota capaz de aceitar arquivos enviados e configurar onde eles devem ser enviados ser salvo.
- Você se pergunta o que é um CGI é? •
Como você não chamará o CGI diretamente, use o caminho completo como PATH_INFO.
- Lembre-se apenas de que, para uma solicitação em partes, seu servidor precisa desfazê-la, o CGI esperará EOF como fim do corpo. • As mesmas coisas para a saída do CGI. Se nenhum content_length for retornado do CGI, EOF marcará o fim dos dados retornados.
- Seu programa deve chamar o CGI com o arquivo solicitado como primeiro argumento.
- O CGI deve ser executado no diretório correto para acesso ao arquivo de caminho relativo. • Seu servidor deve funcionar com um CGI (php-CGI, Python e assim por diante).

Você deve fornecer alguns arquivos de configuração e arquivos básicos padrão para testar e demonstrar que cada recurso funciona durante a avaliação.



Se você tiver alguma dúvida sobre um comportamento, compare o comportamento do seu programa com o do NGINX.
Por exemplo, verifique como funciona o server_name.
Compartilhamos com vocês um pequeno testador. Não é obrigatório passar se tudo funcionar bem com seu navegador e testes, mas pode ajudar você a caçar alguns bugs.



O importante é a resiliência. Seu servidor nunca deve morrer.



Não teste com apenas um programa. Escreva seus testes com uma linguagem mais conveniente, como Python ou Golang, e assim por diante. Até mesmo em C ou C++, se quiser.

Capítulo IV

Parte bônus

Aqui estão os recursos extras que você pode adicionar:

- Suporte a cookies e gerenciamento de sessão (prepare exemplos rápidos).
- Lidar com múltiplos CGI.



A parte bônus só será avaliada se a parte obrigatória for PERFEITA. Perfeito significa que a parte obrigatória foi feita integralmente e funciona sem mau funcionamento. Se você não passou em TODOS os requisitos obrigatórios, sua parte bônus não será avaliada.

Capítulo V

Submissão e avaliação por pares

Entregue sua tarefa no seu repositório Git como de costume. Apenas o trabalho dentro do seu repositório será avaliado durante a defesa. Não hesite em verificar novamente os nomes dos seus arquivos para garantir que estejam corretos.



16D85ACC441674FBA2DF65190663F42A3832CEA21E024516795E1223BBA77916734D1
26120A16827E1B16612137E59ECD492E46EAB67D109B142D49054A7C281404901890F
619D682524F5