EA872 - Laboratório de Programação de Software Básico

Atividade 6

Implementação de funções básicas de servidor HTTP

Introdução

Esta atividade objetiva criar as funções básicas necessárias para um servidor http. O programa obtido irá interagir apenas com arquivos nesta etapa, operando com requisições e respostas em disco. Nas próximas etapas, faremos o servidor operar diretamente com a rede.

Todas as decisões de projeto devem estar bem documentadas no relatório da atividade. Os programas implementados também deverão estar devidamente documentados com comentários dentro do código fonte, além de explicações sobre seu funcionamento e sobre os resultados de seus testes.

Atividades em sala de aula

1 Crie uma estrutura de diretórios para ser o seu webspace.

`meu-webspace` (diretório com permissão de leitura, escrita e execução para owner)

`meu-webspace/index.html` (arquivo com permissão de leitura para owner)

`meu-webspace/dir1` (diretório dir1 com permissão de leitura, escrita e execução para owner)

`meu-webspace/dir1/dir11` (diretório dir11 com permissão de leitura, escrita e execução para owner)

`meu-webspace/dir1/texto1.html` (arquivo texto1.html com permissão de leitura para owner)

`meu-webspace/dir1/texto2.html` (arquivo texto2.html sem permissão de leitura para owner)

`meu-webspace/dir2` (diretório com permissão de leitura e escrita, mas sem permissão de execução para owner)

Os conteúdos dos arquivos html podem ser qualquer coisa de sua escolha. Veja abaixo o conteúdo mínimo de uma página html que você pode usar para criar suas páginas. Sugerimos usar conteúdos e títulos diferentes para cada uma de maneira a facilitar sua identificação.

```
<!DOCTYPE html>
<html>
      <head>
           <title>Este é o título da página. Altere como quiser.</title>
       </head>
       <body>
           Aqui está o conteúdo da página. Altere-o de maneira a
           facilitar a identificação de cada página. Ajudaria dizer aqui
           qual é a página e onde ela está na estrutura de diretórios.
      </body>
</html>
```

1.1 Documente em seu relatório de atividades em sala a estrutura do webspace, usando o comando `ls -lR meu-webspace`.

1.2 Documente em seu relatório de atividades em sala o conteúdo de cada um dos arquivos html criados.

2 Precisamos criar agora um conjunto de requisições web bem completas como já feito em atividades anteriores. Para isso vamos gravar as requisições que os navegadores enviam. A fim de obter tais requisições completas e bem formatadas, podemos alterar o programa `http-dump` (Atividade 4) de modo que o mesmo salve as requisições de um ou mais navegadores comerciais em arquivos em disco. Estes arquivos serão usados como requisições de entrada do servidor web implementado nesta atividade.

Obs: em lugar de alterar o `http_dump.c`, podemos simplesmente redirecionar a saída do mesmo para um arquivo de saída, usando o operador de shell ">"; neste caso será preciso editar tal arquivo de saída para eliminar linhas extras que não tenham relação com a requisição HTTP.

Usando o `http-dump-modificado` ou o recurso de redirecionamento seguido de edição dos arquivos resultantes, obtenha um arquivo de requisição GET para buscar cada recurso descrito a seguir:

- diretório `dir1` existente e com permissão de execução na raiz do seu webspace;
- diretório `dir11` existente e com permissão de execução dentro do diretório `dir1` (`dir11` deve estar vazio);
- arquivo `texto1.html` existente e com permissão de leitura no `dir1` ;
- arquivo `texto2.html` existente e sem permissão de leitura em `dir1`;
- diretório `dir2` existente e sem permissão de execução na raiz do seu webspace;

Obs:

- Use um arquivo de requisição para cada uma destas 5 requisições GET diferentes.
- Dependendo do tipo e da versão de seu navegador, pode ser que ele envie requisições GET extras, devido ao fato de `http-dump` não responder e/ou porque ele está tentando obter ícones em formato GIF, que são opcionais e não precisam estar presentes em um servidor padrão. Você deve remover tais requisições extras dos arquivos.
- Cada arquivo de requisição HTTP deverá ter informações referentes a apenas uma requisição.
- O protocolo HTTP 1.1 documentado na RFC9110 determina que todas as linhas devem ser terminadas pelos caracteres não-imprimíveis `CR` (Carriage Return) e `LF` (Line Feed). Portanto, as requisições colhidas de navegadores deverão estar seguindo esta determinação, o que faz com que haja mais caracteres do que é visto em cada linha do arquivo coletado. Use o comando `cat -A filename` em Linux para comprovar a existência dos caracteres não imprimíveis de fim de linha (`^M` e `$`).

2.1 Documente no relatório com `cat -A` o conteúdo de cada requisição

coletada.

2.2 Explique por que há linhas terminadas em `^M$` e outras só em `$` em um mesmo arquivo.

3 As requisições que não podem ser enviadas livremente a partir de um browser (caso de OPTIONS, HEAD e TRACE) devem ser criadas com um editor de texto, de acordo com a sintaxe definida na RFC9110. Neste caso, os dois caracteres especiais (`CR` e `LF`) não estarão presentes (se o arquivo foi criado em ambiente Unix). Uma forma de inseri-los é editar tal arquivo em ambiente Windows ou usar um editor (em hexadecimal) que permita introduzir estes caracteres de controle diretamente. Uma outra forma é editar um dos arquivo com requisição GET que o `http-dump` forneceu para transformá-la em outra requisição, aproveitando os caracteres especiais (`CRLF`) já presentes. Lembre-se que uma requisição HTTP sempre tem que terminar com uma linha em branco.

3.1 Documente, usando `cat -A`, cada um dos arquivos com as requisições HEAD feitas para cada item da estrutura do seu webspace.

3.2 Documente, usando `cat -A`, cada um dos arquivos com as requisições OPTIONS e TRACE.

Atividades para a próxima semana

4 A função criada na Atividade 5 já está bem próxima de uma função para processar requisições GET vindas de um browser. Ela também pode ser usada como base para uma função dedicada à requisição HEAD. Portanto faça as adaptações necessárias no código da Ativ. 5 e crie as funções específicas para atender a requisições GET e HEAD. O recurso retornado por estas funções deve ser acompanhado de um cabeçalho formal, seguindo rigorosamente o formato das mensagens de retorno definidas na RFC9110 e observadas na Atividade 4. O valor do parâmetro `Date` do cabeçalho deverá refletir a data e hora reais, as quais podem ser obtidas e formatadas por chamadas de sistema tais como `gettimeofday()` e `asctime()`. Os valores dos parâmetros `Content-Length` e `Last-Modified` poderão ser obtidos pela chamada `fstat()`.

O cabeçalho de cada resposta deverá ter pelo menos os seguintes campos:

- código de resposta HTTP (200, 300, 400 etc) – ex: `HTTP/1.1 200 OK`
- data da resposta – ex: `Date: Fri Sep 1 15:20:12 2023 BRT`
- tipo de servidor – ex: `Server: Servidor HTTP ver. 0.1 de José da Silva`
- procedimento com a conexão – ex: `Connection: close` ou `keep-alive` (usar o que o browser forneceu na requisição)
- data de última alteração do recurso solicitado – ex: `Last-Modified: Fri Sep 1 14:50:54 2023 BRT`
- tamanho do recurso (em bytes) – ex: `Content-Length: 116`
- tipo de conteúdo – ex: `Content-Type: text/html` ou `text/plain` ou `image/jpeg`

etc (nesta atividade iremos nos limitar a arquivos html, mas é importante já ir pensando na possibilidade de devolver recursos como imagens, vídeos, áudios etc.).

5 Complemente este repertório de funções, criando outras para processar requisições TRACE e OPTIONS (lembre-se de que estas duas não estão ligadas a um recurso em particular, mas precisam de um cabeçalho de retorno também).

6 Crie também uma função para processar todos os tipos de erros (arquivos não encontrados, acesso negado, etc.), a qual deve gerar mensagens em html informando o usuário a respeito do erro ocorrido. Os tipos de erro possíveis podem ser encontrados na RFC9110.

7 Integre todas as funções ao parser que ficou pronto na Atividade 4. Esta integração completa a versão inicial de nosso servidor. Ele deverá obter uma requisição HTTP de um arquivo em disco (req_N.txt), imprimir a requisição na tela e enviar o resultado do processamento da requisição para outro arquivo em disco (resp_N.txt). A forma de uso desta versão inicial do servidor deverá ser a seguinte:

```
./servidor <Web Space> req_N.txt resp_N.txt registro.txt
```

onde:

- servidor é o nome do programa que você compilou,
- <Web Space> é o caminho para a raiz do espaço web escolhido,
- req_N.txt é o N-ésimo arquivo com uma requisição http,
- resp_N.txt é o arquivo que receberá a resposta completa (cabeçalho mais conteúdo html/jpeg/gif etc) para a N-ésima requisição e
- registro.txt é um arquivo que conterá cópias das requisições, sendo cada cópia seguida da resposta (mas só o cabeçalho da mesma, sem o conteúdo do recursos solicitado, no caso de GET);

- Obs:
  - as requisições lidas dos arquivos de entrada (req_N.txt) deverão ser completamente analisadas pelo parser e armazenadas na estrutura de dados especificada na Atividade 4;
  - é possível fazer com que o parser leia de um arquivo e não da entrada padrão mudando o conteúdo do descritor de entrada yyin;
  - alternativamente é possível fazer com que o parser leia de um buffer na memória (que contenha o que foi lido de um arquivo previamente) usando a chamada yy_scan_string();
  - a impressão dos detalhes das requisições na tela deverá ser feita a partir de uma varredura do conteúdo desta estrutura de dados (como feito na atividade 4);
  - o parser deverá considerar a questão das linhas terminadas em `CRLF` para poder funcionar adequadamente;
  - cada novo par requisição/resposta deverá ser anexado ao final do arquivo registro.txt e deverá estar bem separado dos demais com espaços e/ou delimitadores para facilitar sua identificação.

8 O relatório de atividades deve documentar e explicar detalhadamente:

8.1 as funções criadas

8.2 os resultados dos testes feitos com cada uma das funções, usando os diversos arquivos de entrada req_N.txt . Todas as funções devem ser testadas, incluindo a de tratamento de erros em suas diversas possibilidades. Todos os testes deverão ser documentados.

- Obs: se a página html enviada em resposta a uma requisição GET for muito extensa, ela deverá ser truncada no relatório.

A figura abaixo mostra a estrutura esperada nesta versão do servidor web em desenvolvimento.