Faculdade de Engenharia Elétrica e de Computação – Laboratório de Programação de Software Básico (EA872)

Trabalho final: Programa servidor HTTP

Aluno: Mateus Henrique Silva Araújo
RA: 184940
Turma: K
Data: 01/12/2023

Sumário

1) Descrição do sistema desenvolvido:

1.1) Introdução e objetivos:

Durante o semestre, com intuito de trabalhar-se os conceitos associados a programação de software básico, como geração e operacionalização de compiladores, gerenciamento de estrutura de arquivos, multiprogramação e threads, foram realizadas diversas atividades práticas as quais, gradualmente, levaram à implementação de um aplicativo que atua como um servidor HTTP capaz de se comunicar pela rede com clientes e responder suas requisições.

Nesse sentido, o objetivo desta atividade é aglutinar todos os conhecimentos obtidos até então para se gerar uma versão final do aplicativo servidor, a qual não só deve ser capaz de validar todas as especificações estabelecidas, mas também operar de forma eficiente, confiável e veloz.

1.2) Especificações:

Assim como indicado no roteiro da atividade final, o conjunto mínimo de funcionalidades que o aplicativo servidor deve descrever é o seguinte: capacidade de ser executado e receber parâmetros por meio do terminal; capacidade de receber requisições HTTP por meio da comunicação via rede utilizando-se de soquetes, processá-las diretamente por meio do código gerado pelas ferramentas *flex* e *bison*, e respondê-las conforme o regulamento desse tipo de comunicação; bem como implementar um fluxo de execução eficiente, usando-se, para tal, de threads que atuam em aparente paralelismo no sistema.

Dessa forma, focando sobre a comunicação entre cliente e servidor, é estabelecido que o aplicativo deve ser capaz de responder requisições para os métodos GET, HEAD, OPTIONS, POST e TRACE, enviando, quando necessário, páginas *html* que indiquem, apropriadamente, algum erro que tenha ocorrido durante a resposta de uma requisição. Além disso, também é especificado que deve ser desenvolvido um mecanismo o qual permita, mediante o envio de formulários usando se de requisições com método POST, a alteração de arquivos de senhas empregues para eventuais configurações de segurança de recursos roteados.

Por fim, no quesito segurança, é definido que o servidor deve confinar a busca e requisição de arquivos por um cliente ao diretório no qual se encontra o *web-space* (impedindo, portanto, que informações acerca de outros arquivos fora deste sejam enviadas como resposta). Ademais, também deve ser desenvolvida a capacidade de se proteger recursos por meio do uso de arquivos *.htaccess*, que apontam para arquivos externos ao espaço roteado, nos quais cada linha contém um par "usuário : senha criptografada". Tais senhas devem ser cifradas usando-se a chamada de sistema `crypt()` com *salt* "EA" e o mecanismo de autenticação usado para o acesso desses recursos deve ser do tipo *Basic Authentication*.

1.3) Estrutura da aplicação:

Dado a complexidade e extensão do código gerado, foi necessário, com intuito de elevar sua legibilidade e organização, segmentá-lo em diferentes módulos e separar os vários arquivos produzidos em diretórios característicos. Dessa forma, assim como é apresentado na figura 1, quatro diretórios principais foram criados na raiz do diretório da aplicação: "src", "include", "parser" e "html".

O primeiro armazena os arquivos de código C usados na compilação do programa, o segundo armazena os arquivos de cabeçalho (.h) usados pelos arquivos .c, o terceiro armazena os arquivos fontes para a geração dos analisadores léxico e sintático (.l e .y) por meio das ferramentas *flex* e *bison* e o quarto armazena as páginas *html* de erros e avisos padrões que o servidor pode enviar ao cliente. A composição interna de cada um desses diretórios secundários também é mostrada na figura 1.

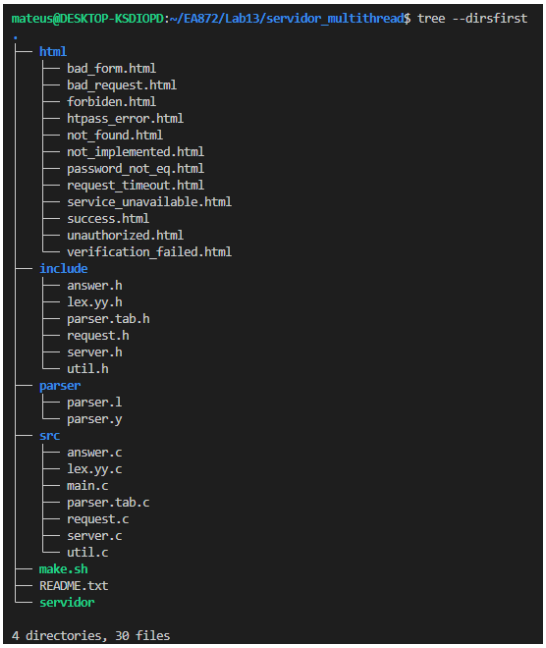


Figura 1: *composição do diretório raiz da aplicação e de seus diretórios secundários.*

A modularização realizada sobre o código-fonte nada mais é que a separação do programa principal em vários arquivos de código e cabeçalho de modo a agrupar partes coerentes que produzem uma mesma funcionalidade. Dessa forma, assim como é possível notar nas figuras acima, o programa foi dividido em 6 módulos distintos:

- **Módulo `server`**: engloba os arquivos "server.c" e "server.h", sendo responsável por realizar a configuração do servidor e operar as comunicações com os clientes. Nele se encontram tanto o fluxo principal de execução no qual a thread original se mantém, bem como a função para qual as threads produzidas pela original são enviadas para o tratamento de uma conexão específica;
- **Módulo `answer`:** engloba os arquivos "answer.c" e "answer.h", sendo responsável por implementar o fluxo de geração de uma mensagem de resposta do servidor. Nele se encontram funções que geram os cabeçalhos da mensagem de resposta, verificam a disponibilidade e necessidade de autorização de recursos, bem como processam possíveis erros;
- **Módulo `request`:** engloba os arquivos "request.c" e "request.h", no qual estão agrupadas funções associadas ao processamento de uma requisição recebida. É nele que se encontram os métodos invocados no código do analisador sintático para a construção da lista ligada que armazena os cabeçalhos e parâmetros da requisição. Além disso, nesse módulo está presente a função que configura as estruturas e ativa o *parser* reentrante;
- **Módulo `util`:** engloba os arquivos "util.c" e "util.h", sendo um módulo utilitário que disponibiliza funções de uso geral, como para manipulação de *strings* ou realização de codificações, que são empregues em todos os demais módulos.
- **Módulo `parser`:** esse módulo está relacionado com os arquivos gerados pelas ferramentas *flex* ("lex.yy.c" e "lex.yy.h") e *bison* ("parser.tab.c" e "parser.tab.h"), os quais implementam o analisador léxico e sintático que, com o apoio das funções disponibilizadas pelo módulo `request`, transforma a requisição enviada pelo cliente em uma lista ligada de cabeçalhos e seus parâmetros;
- **Módulo `main`:** é o módulo mais simples, composto somente pelo arquivo "main.c". Sua responsabilidade é realizar verificações sobre os parâmetros de entrada do programa, de modo a abortar seu funcionamento se algum deles for inválido ou chamar a função que executa o servidor (do módulo `server`) caso contrário.

Maiores explicações sobre o funcionamento, bem como a interação, de cada um desses blocos serão fornecidas na seção 2.

1.4) Funcionalidades implementadas e não implementadas:

O programa desenvolvido, ao ser executado, obtém de sua linha de comando os parâmetros que correspondem ao caminho para o *web-space* roteado, a porta na qual o servidor deverá atuar, o número máximo de threads que podem operar no sistema e a codificação dos arquivos de texto salvo no *web-space*. Ele se comunica com clientes por meio da conexão via rede, lendo requisições do soquete e fornecendo-as diretamente para seus analisadores léxico e sintático reentrantes (os quais foram gerados com o uso das ferramentas estabelecidas). O procedimento de leitura, geração e envio de resposta a um cliente é completamente executado por uma thread secundária gerada pela thread principal, a qual se ocupa somente de aceitar conexões.

Além disso, o servidor é capaz de responder adequadamente a requisições dos métodos GET, HEAD, OPTIONS, TRACE e POST, podendo enviar arquivos do tipo *txt*, *html*, *pdf*,

jpeg, *png*, *gif* e *tif* em tal resposta (apresentando, também, uma grande capacidade de extensão desses tipos). Durante esse processo, ele é capaz de capturar um eventual erro e discernir entre os tipos possíveis, enviando uma página *html* característica ao cliente.

Em questão de segurança, ele realiza o procedimento de autenticação *Basic Authentication* quando um recurso apresenta proteção, ou seja, há um arquivo *.htaccess* em um diretório no caminho até o recurso que aponta para um arquivo contendo pares usuário e senha criptografada. Ele também fornece, por meio de formulários enviados em requisições de método POST, a possibilidade de se alterar a senha de um usuário presente em um destes arquivos de senhas, mantendo-se a formatação especificada, isto é, mantendo as senhas salvas criptografadas usando-se de um *salt* obtido da própria senha anterior. Ademais, o servidor impede que qualquer recurso fora do *web-space* roteado seja acessado por meio da verificação da profundidade relativa do pedido.

Dessa forma, conclui-se que foram implementadas não só todas as especificações mínimas, mas também algumas capacidades adicionais (como o desenvolvimento de analisadores reentrantes, que permitem o uso destas ferramentas paralelamente por threads distintas sem que uma atrapalhe o resultado da outra, tornado o programa mais eficiente). Entretanto, analisando os outros recursos que foram requisitados nas atividades anteriores e que não foram indicados como obrigatórios, destaca-se a ausência da produção de um arquivo de *log* no qual seriam registrados os cabeçalhos de cada requisição recebida e da resposta enviada pelo servidor. Essa ausência foi uma escolha tomada com intuito de aumentar a eficiência de processamento multithread da aplicação, ao passo que, como duas threads diferentes não poderiam abrir e escrever neste arquivo ao mesmo tempo, este acesso passaria a ser uma região crítica que eventualmente travaria threads em seu semáforo.

1.5) Manual de uso do sistema:

Para poder utilizar o sistema desenvolvido, primeiramente, execute o *script* “make.sh”, presente no diretório raiz do projeto, para que este compile a versão mais atual do programa. Em seguida, o arquivo executável “servidor” terá sido gerado na raiz, então basta fornecer uma linha de comando do tipo: `./servidor <meu web-space> <porta> <número máximo de threads> <charset do web-space>`, onde o primeiro campo deve ser substituído pelo caminho até a raiz do diretório que se deseja rotear como *web-space*; o segundo , pelo número da porta que o servidor deve utilizar; o terceiro, pelo número máximo de threads que operarão no sistema e o quarto, pela codificação dos arquivos texto do *web-space*.

Vale lembrar que, caso seja fornecido algum valor inadequado para os três primeiros campos, o programa alertará o problema e encerrará sua execução. Entretanto, não é feita nenhuma verificação do valor fornecido no quarto campo (haja vista a elevada quantidade de codificações disponíveis). Isto implica que, se o usuário fornecer um valor incorreto, os arquivos de texto enviados pelo servidor muito provavelmente estarão errados.

2) Diagramas do sistema:

2.1) Fluxograma do sistema:

Para facilitar a explicitação do funcionamento do sistema, um fluxograma na forma de diagrama de atividades UML foi desenvolvido e está apresentado (devido ao seu tamanho) no primeiro apêndice, ao final deste documento. Este diagrama tem como intuito esboçar o funcionamento geral do sistema, enfatizando os principais procedimentos que ocorrem nele.

Dessa forma, nem todas as funções usadas em cada módulo tiveram sua lógica explicitada no diagrama em questão. Entretanto, como o código-fonte foi exaustivamente comentado, espera-se que, ao usar do diagrama para acompanhar o fluxo de funcionamento do programa e recorrer ao código-fonte para maiores detalhes, seja possível compreender os procedimentos adotados com clareza.

2.2) Diagrama de componentes:

Com o objetivo de não só tornar mais clara a interação entre os diferentes módulos do programa, mas também indicar explicitamente as funções que promovem a interligação entre eles, o diagrama de componentes apresentado na figura 2 foi desenvolvido.

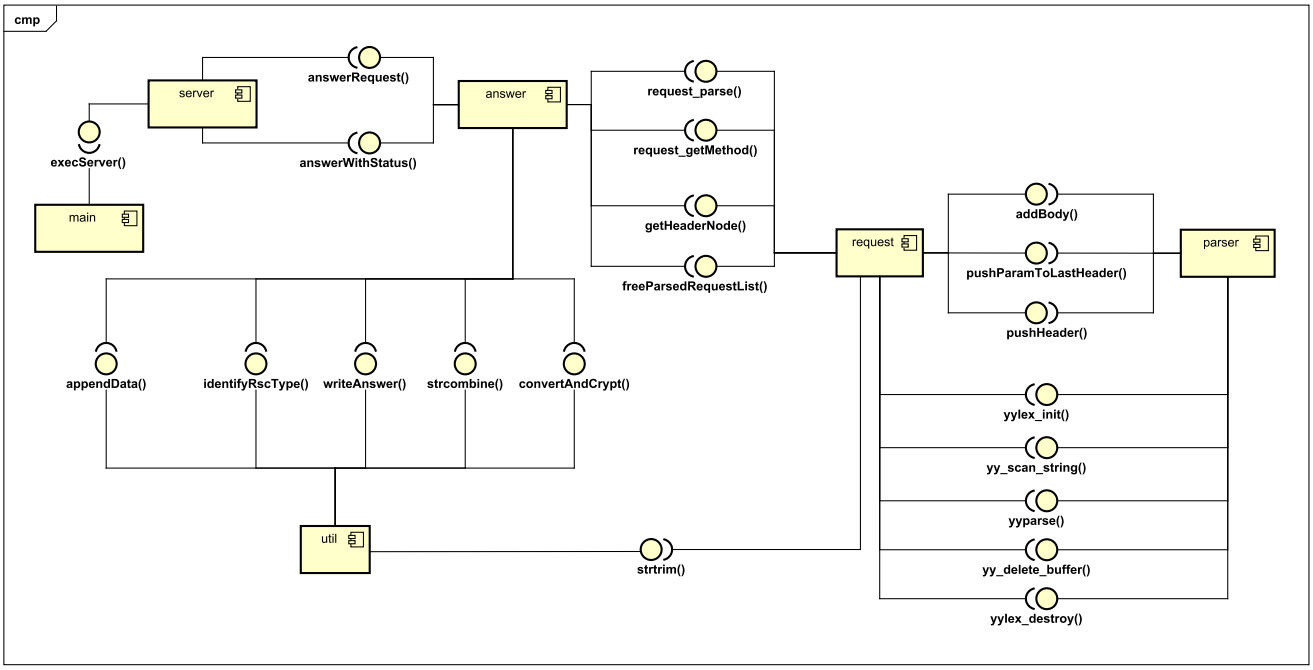


Figura 2: *Diagrama de componentes do aplicativo servidor. Nele, cada interface provida representa uma função implementada pelo módulo provedor e oferecida para uso externo, bem como cada interface requerida representa uma função de outro módulo que o módulo requerente faz uso.*

Dessa forma, analisando o diagrama acima, nota-se que o módulo `main` se conecta unicamente com o módulo `server`, fazendo isso por meio da função `execServer()`, cuja responsabilidade é implementar o fluxo principal do funcionamento do servidor. Esta função recebe como argumentos os mesmos parâmetros passados pela linha de comando na chamada do executável e cuja validade foi certificada no módulo `main`, isto é: o caminho para o diretório do *web-space* roteado, o número da porta na qual o servidor deverá operar, o número máximo de threads ativas no sistema e a codificação dos arquivos de texto roteados.

Em seguida, percebe-se que o módulo `server` se conecta com o módulo answer por meio das funções `answerRequest()` e `answerWithStatus()`. Tais funções possuem comportamento parecidos, tendo a responsabilidade de gerar uma resposta para o cliente e escrevê-la no soquete de comunicação. A diferença fundamental entre essas duas funções é que a segunda é executada quando ocorre algum erro de comunicação (como um timeout na espera de uma mensagem ou superlotação na quantidade de conexões), fato que implica no conhecimento prévio do status da resposta e faz com que seja desnecessária uma requisição. Dessa forma, em sua chamada são fornecidos somente o descritor que aponta para o soquete de comunicação com o cliente e o status pré-definido

para a resposta. Em contrapartida, a primeira função é usada para se gerar as respostas usuais às requisições de clientes, fazendo-se necessário o fornecimento, além do descritor do soquete, de uma *string* contendo a requisição enviada.

Quanto ao módulo `answer`, nota-se que há conexões tanto com o módulo `util`, para uso das funções de utilidade disponibilizadas, quanto com o módulo request, para o uso da função que gera a análise da *string* contendo a requisição (função `request_parse()`) e produz a lista ligada de cabeçalhos e parâmetros, das funções que obtém nós e informações específicas desta lista (funções `request_getMethod()`, `getHeaderNode()`), bem como da função que desaloca toda a memória alocada nela (função `freeParsedRequestList()`).

Por fim, destaca-se a interação direta entre os módulos `request` e `parser`. Nesta relação, o módulo `parser` fornece as funções que tanto produzem as estruturas que implementam o analisador reentrante (`yylex_init()`), quanto define sua *string* alvo (`yy_scan_string()`), bem como o ativam (`yyparse()`) e, em seguida, desalocam a memória usada por ele (`yy_delete_buffer()` e `yylex_destroy()`). Já o módulo `request` fornece as funções que são chamadas no arquivo "parser.y", as quais constroem a lista ligada de cabeçalhos e parâmetros conforme a análise progride.

Novamente recomenda-se a leitura dos vários comentários adicionados no corpo das funções mencionadas acima para a melhor compreensão de suas lógicas de operação.

3) Testes de sistema:

3.1) Estrutura de *web-space* criada para teste:

Para poder testar as funcionalidades essenciais exigidas, o *web-space* fornecido como base para testes foi estendido de modo a adicionar os arquivos *.htaccess* faltantes, gerando-se também arquivos de senhas diferentes para cada um deles. Nesse sentido, a figura 6 apresenta a estrutura de diretórios final usada para todos os testes da aplicação, a figura 3 e 4 expõem o conteúdo de cada arquivo *.htaccess* presente no *web-space* e de seus arquivos de senhas correspondentes, bem como a figura 5 apresenta uma relação entre os usuários presentes em cada arquivo de senhas e os valores descriptografados que correspondem às senhas destes.

```
mateus@DESKTOP-KSDIOPD:~/EA872/Lab13/webspace_para_testes$ cat dir1/dir11/.htaccess -
../senhas_dir11.txt

mateus@DESKTOP-KSDIOPD:~/EA872/Lab13/webspace_para_testes$ cat dir1/dir11/dir111/.htaccess -
../senhas_dir111.txt

mateus@DESKTOP-KSDIOPD:~/EA872/Lab13/webspace_para_testes$ cat dir4/dir41/.htaccess -
../senhas_dir41.txt
```

Figura 3: *conteúdo dos arquivos .htaccess adicionados ao web-space base para testes.*

```
mateus@DESKTOP-KSDIOPD:~/EA872/Lab13/servidor_multithread$ cat ../senhas_dir11.txt -
user5:EAh11UcXydMIM
user3:EA/pH0jmlKfP6
user4:EAn8Hfdrbbtvg
user1:EACxq6nE/7pmw
user2:EAuCql5t/bxq2

mateus@DESKTOP-KSDIOPD:~/EA872/Lab13/servidor_multithread$ cat ../senhas_dir111.txt -
user1:EA1VLGyl9y6F6
user2:EA8/ZTzJ3dC4o
user3:EAcOr7oulatl6


mateus@DESKTOP-KSDIOPD:~/EA872/Lab13/servidor_multithread$ cat ../senhas_dir41.txt -
user2:EA1AkHKq4eyVY
user5:EABWmnk3z9qEE
user1:EArF59IJJcFL6
user3:EAtarOXmxIxTk
user4:EAWPYXYHdCiXA
```

Figura 4: *conteúdo dos arquivos de senhas apontados pelos arquivos .htaccess presentes no web-space de testes.*

mapeamento.csv

nome de usuário	senha em senhas_dir11	senha em senhas_dir111.txt	senha em senhas_dir41.txt
user1	senhates	mamamate	mateus123
user2	testeste	aquiaqui	senha123
user3	aaabbbcc	olamundo	senha321
user4	senhasen	--------	12345678
user5	senhozaz	--------	adcdef12

Figura 5: *relação entre pares nome de usuário e senha (em sua versão descriptografada) que foram inseridos em cada arquivo de senhas usados para proteção do web-space de testes*

```
mateus@DESKTOP-KSDIOPD:~/EA872/Lab13$ tree -c -p -a webspace_para_testes/
[drwxr-xr-x]  webspace_para_testes/
├── [-rw-r--r--]  a.txt
├── [-rw-r--r--]  configura_permissoes.sh
├── [drwxr-xr-x]  dir1
│   ├── [-rw-r--r--]  index.html
│   └── [drwxr-xr-x]  dir11
│       ├── [drwxr-xr-x]  dir111
│       │   ├── [-rw-r--r--]  form.html
│       │   ├── [-rw-r--r--]  .htaccess
│       │   └── [-rw-r--r--]  index.html
│       ├── [-rw-r--r--]  form.html
│       ├── [-rw-r--r--]  .htaccess
│       └── [-rw-r--r--]  index.html
├── [drwxr-xr-x]  dir2
│   └── [-rw-r--r--]  welcome.html
├── [drwxr-xr-x]  dir3
│   ├── [-rw-r--r--]  welcome.html
│   └── [--w-r--r--]  index.html
├── [drwxr-xr-x]  dir4
│   ├── [--w-r--r--]  index.html
│   ├── [--w-r--r--]  welcome.html
│   └── [drwxr-xr-x]  dir41
│       ├── [-rw-r--r--]  form.html
│       ├── [-rw-r--r--]  .htaccess
│       ├── [-rw-r--r--]  index.html
│       └── [drwxr-xr-x]  dir411
│           ├── [-rw-r--r--]  form.html
│           └── [-rw-r--r--]  index.html
├── [drwxr-xr-x]  dir5
│   └── [-rw-r--r--]  texto_para_testes.txt
├── [-rw-r--r--]  .directory
├── [-rw-r--r--]  feec.gif
├── [-rw-r--r--]  lab.zip
├── [-rw-r--r--]  teste.pdf
├── [-rw-r--r--]  unicamp.gif
├── [-rw-r--r--]  unicamp.jpg
├── [-rw-r--r--]  unicamp.png
├── [-rw-r--r--]  unicamp.tif
├── [-rw-r--r--]  welcome.html
├── [drw-r-xr-x]  dir
├── [--w-r--r--]  teste.txt
└── [-rw-r--r--]  index.html

10 directories, 31 files
```

Figura 6: *estrutura de diretórios final do web-space de testes*

3.2) Testes efetuados:

1. Requisição de um arquivo regular existente e legível:

Para realizar esse caso de teste, vamos solicitar pelo browser o recurso “a.txt”. Podemos fazer isto ao clicarmos no link “Arquivo txt” na seção “Arquivos” da página principal do *web-space* de teste. O resultado obtido é apresentado na figura 7.

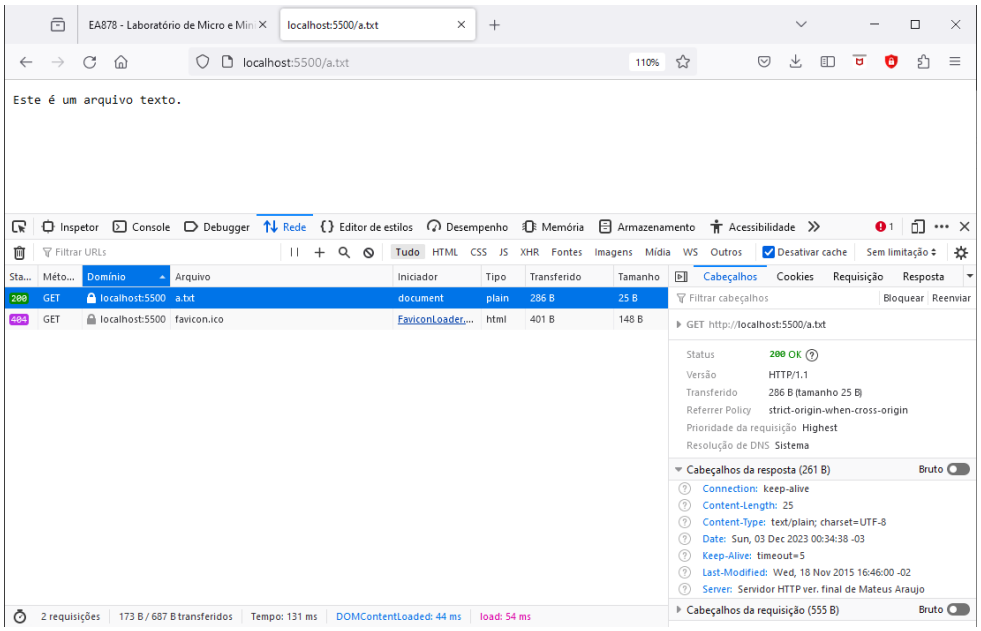


Figura 7: *resultado do teste 1. Nele podemos ver que o arquivo desejado foi enviado adequadamente.*

2. Requisição de um arquivo regular inexistente:

Para realizar esse caso de teste, vamos solicitar pelo browser o recurso “asdfg.html”. Podemos fazer isto ao clicarmos no link “Arquivo inexistente” na seção “Casos especiais” da página principal do *web-space* de teste. O resultado obtido é apresentado na figura 8.

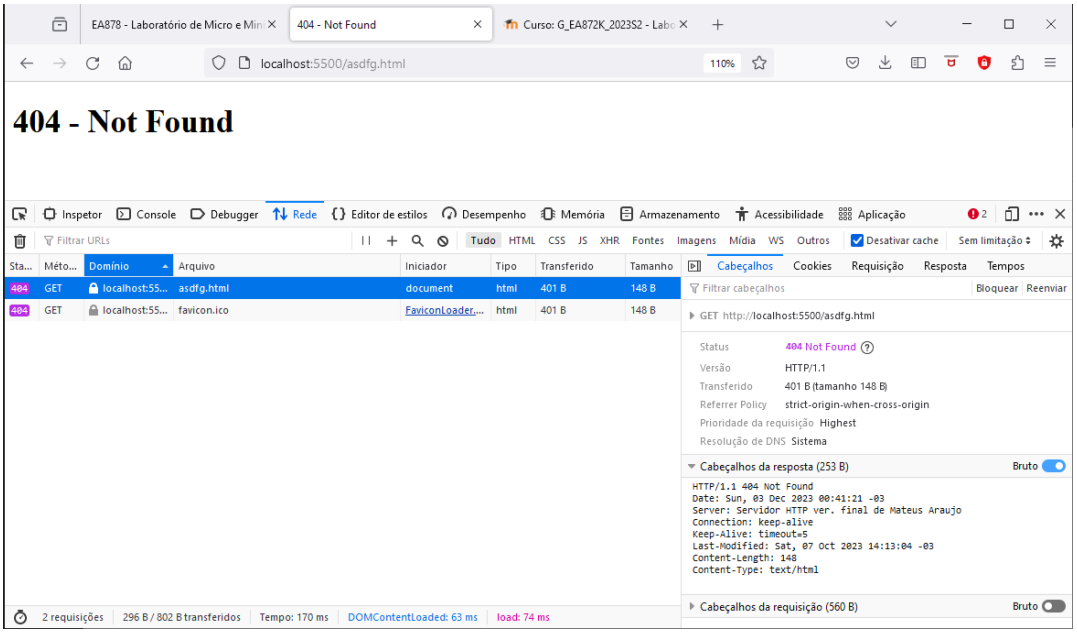


Figura 8: *resultado do teste 2. Nele podemos ver que é retornado o código de erro 404 Not Found, assim como esperado.*

3. Requisição de um arquivo regular existente, porém sem permissão de leitura:

Para realizar esse caso de teste, vamos solicitar pelo browser o recurso “teste.txt”. Podemos fazer isto ao clicarmos no link “Arquivo sem permissão de leitura” na seção “Casos especiais” da página principal do *web-space* de teste. O resultado obtido é apresentado na figura 9.

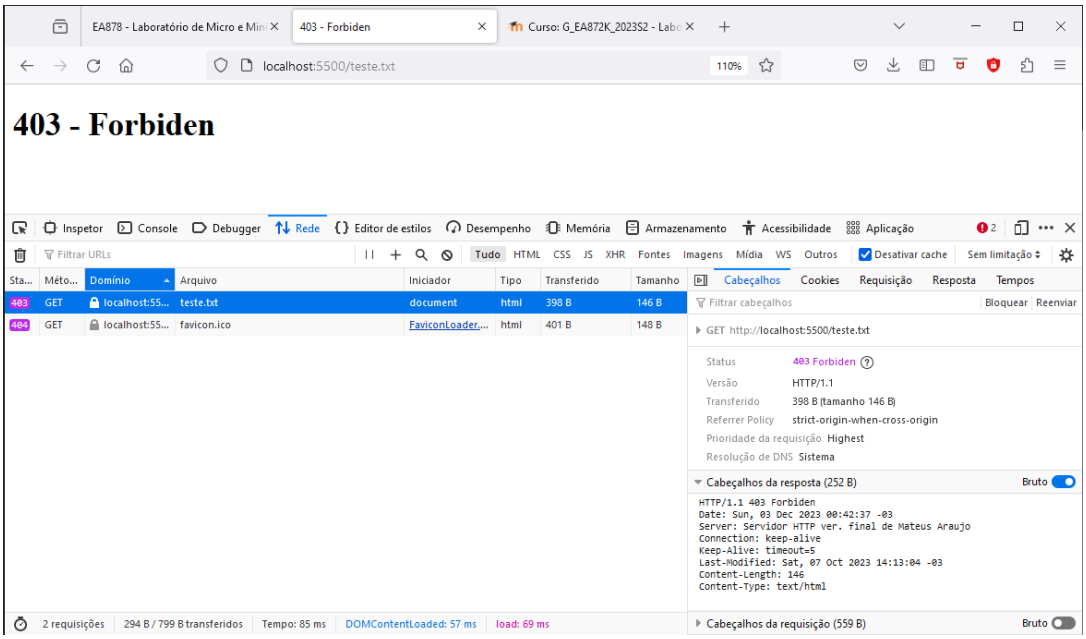


Figura 9: *resultado do teste 3. Nele podemos ver que é retornado o código de erro 403 Forbiden, assim como esperado.*

4. Requisição de um arquivo regular existente, porém dentro de um diretório sem permissão de execução:

Para realizar esse caso de teste, vamos solicitar pelo browser o recurso "dir/texto_de_teste.txt". Podemos fazer isto ao clicarmos no link "Arquivo dentro do diretório acima que está sem permissão de execução" na seção "Casos especiais" da página principal do *web-space* de teste. O resultado obtido é apresentado na figura 10.

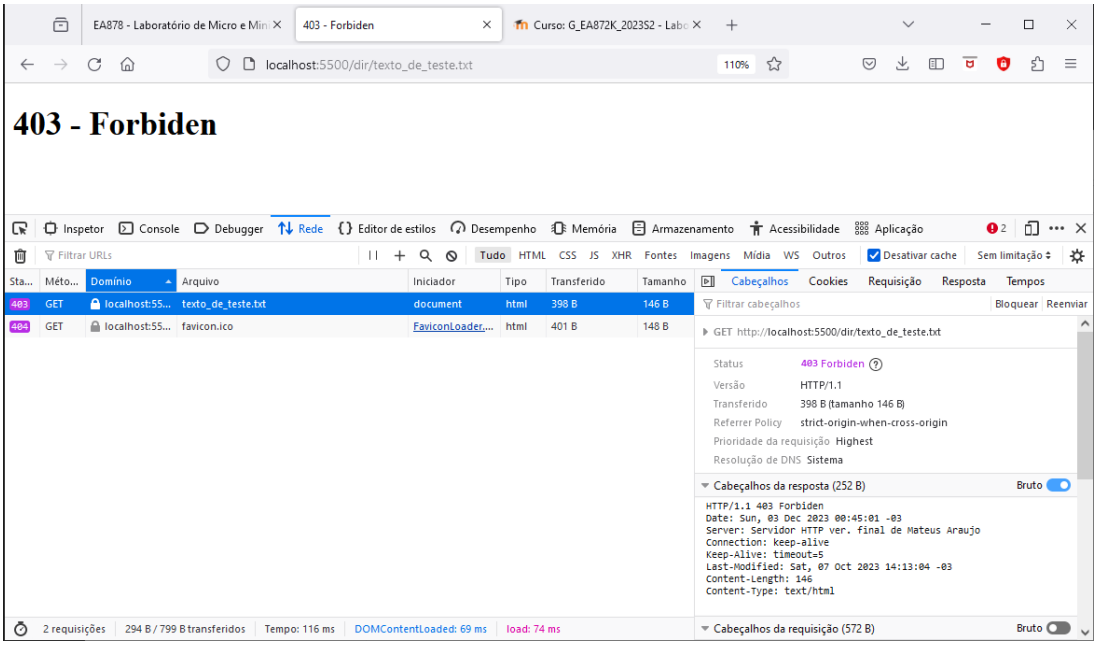


Figura 10: *resultado do teste 4. Nele podemos ver que é retornado o código de erro 403 Forbiden, assim como esperado.*

5. Requisição de um diretório contendo index.html legível:

Para realizar esse caso de teste, vamos solicitar pelo browser o recurso "dir1". Podemos fazer isto ao clicarmos no link "Diretório com index.html" na seção "Casos especiais" da página principal do *web-space* de teste. O resultado obtido é apresentado na figura 11.

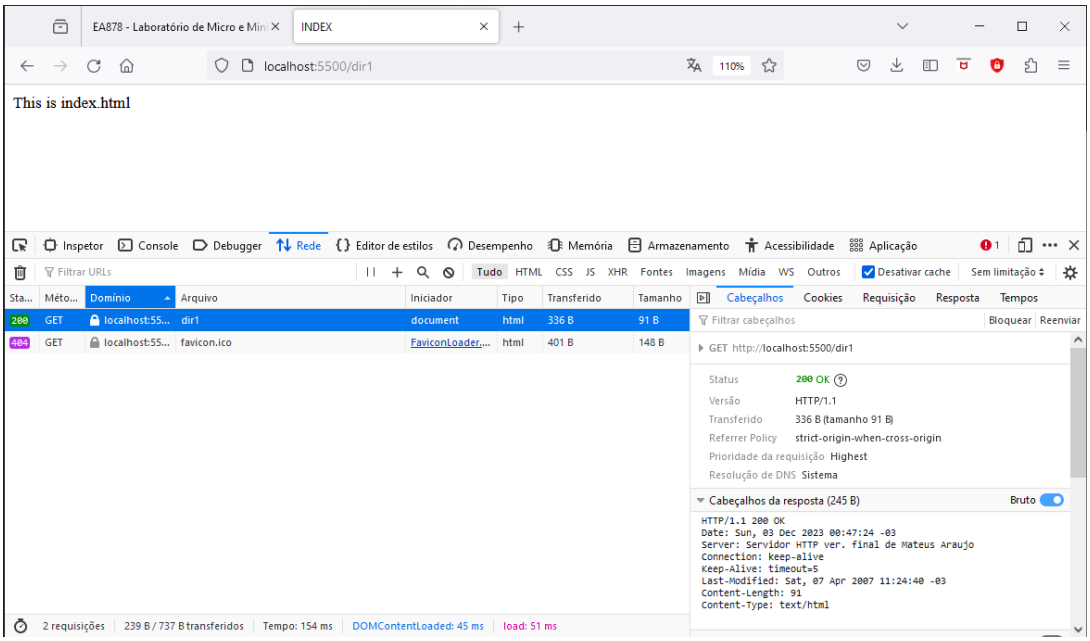


Figura 11: *resultado do teste 5. Nele podemos ver que o arquivo desejado foi enviado adequadamente.*

6. Requisição de um diretório contendo welcome.html legível:

Para realizar esse caso de teste, vamos solicitar pelo browser o recurso “dir2”. Podemos fazer isto ao clicarmos no link “Diretório com welcome.html” na seção “Casos especiais” da página principal do *web-space* de teste. O resultado obtido é apresentado na figura 12.

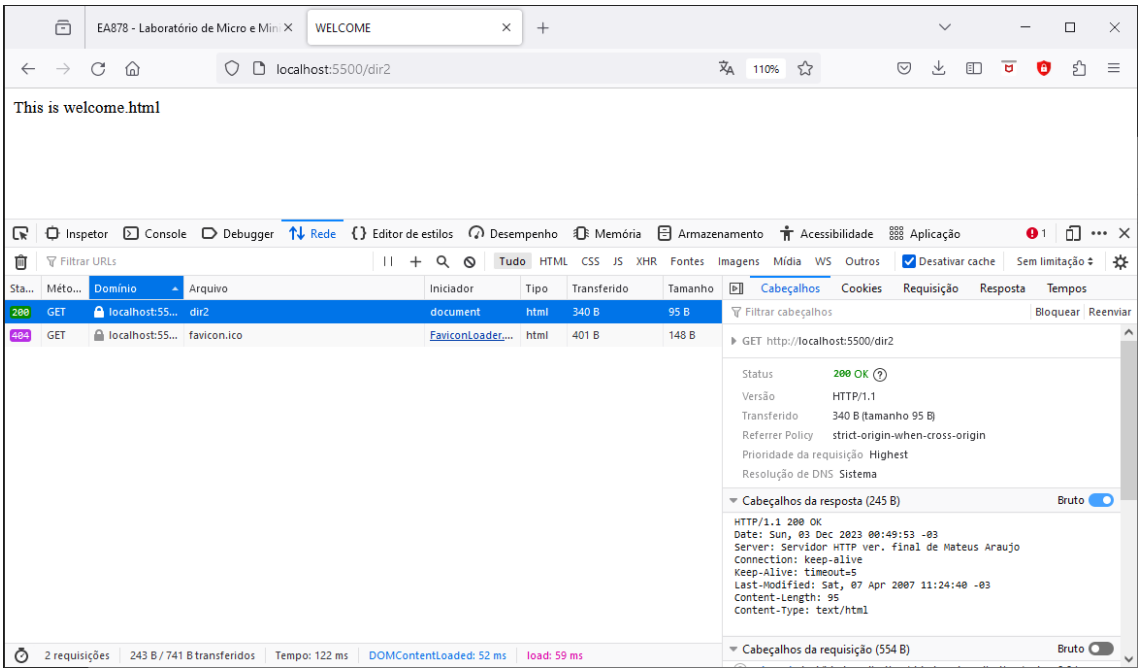


Figura 12: *resultado do teste 6. Nele podemos ver que o arquivo desejado foi enviado adequadamente.*

7. Requisição de um diretório contendo index.html sem permissão de leitura, mas com welcome.html legível:

Para realizar esse caso de teste, vamos solicitar pelo browser o recurso “dir3”. Podemos fazer isto ao clicarmos no link “Diretório com index.html sem permissão de leitura, mas com welcome.html legível.” na seção “Casos especiais” da página principal do *web-space* de teste. O resultado obtido é apresentado na figura 13.

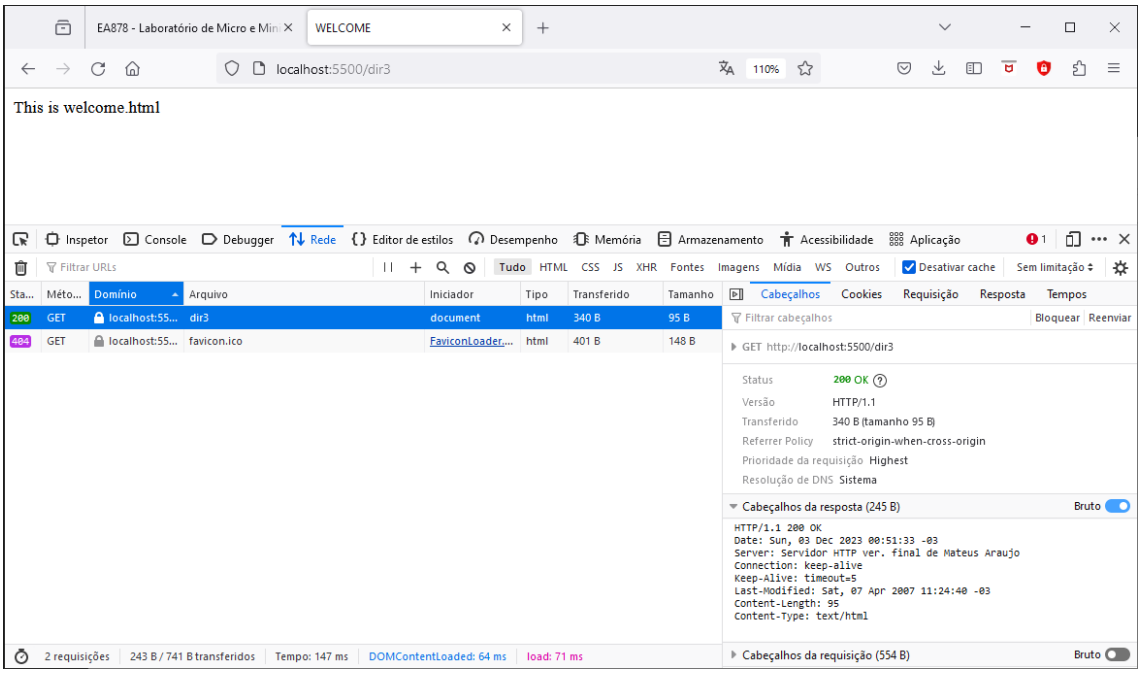


Figura 13: *resultado do teste 7. Nele podemos ver que o arquivo desejado foi enviado adequadamente.*

8. Requisição de diretório contendo index.html e welcome.html ilegíveis:

Para realizar esse caso de teste, vamos solicitar pelo browser o recurso "dir4". Podemos fazer isto ao clicarmos no link "Diretório com index.html e welcome.html sem permissão de leitura" na seção "Casos especiais" da página principal do *web-space* de teste. O resultado obtido é apresentado na figura 14.

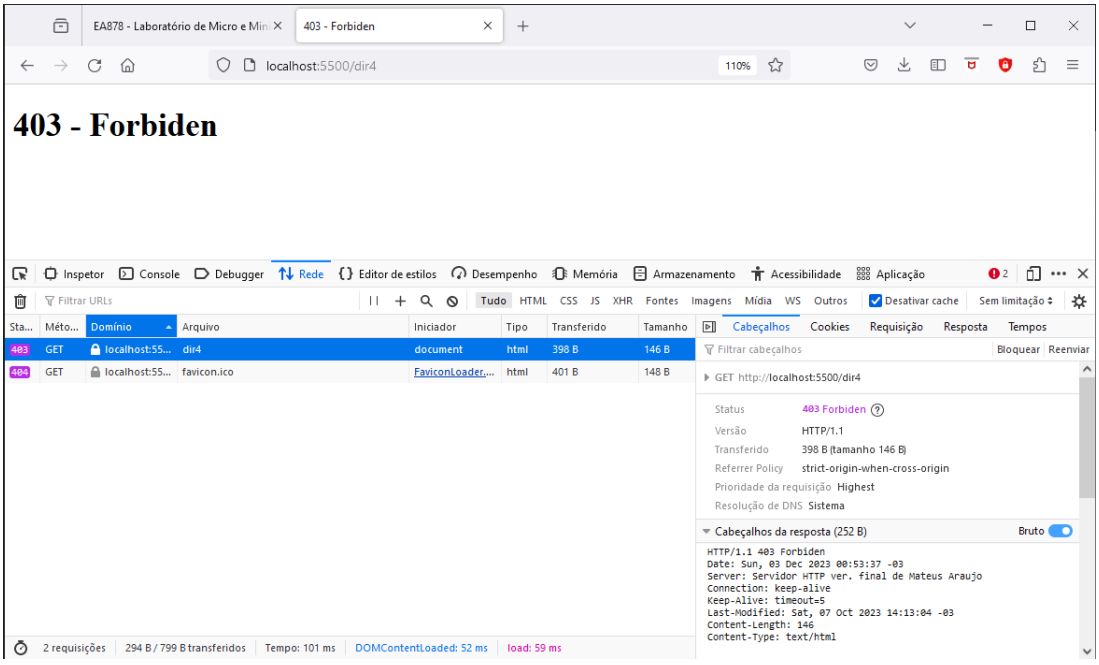


Figura 14: *resultado do teste 8. Nele podemos ver que é retornado o código de erro 403 Forbiden, assim como esperado.*

9. Requisição de um diretório sem páginas padrões:

Para realizar esse caso de teste, vamos solicitar pelo browser o recurso "dir5". Podemos fazer isto ao clicarmos no link "Diretório sem index ou welcome" na seção "Casos

especiais” da página principal do *web-space* de teste. O resultado obtido é apresentado na figura 15.

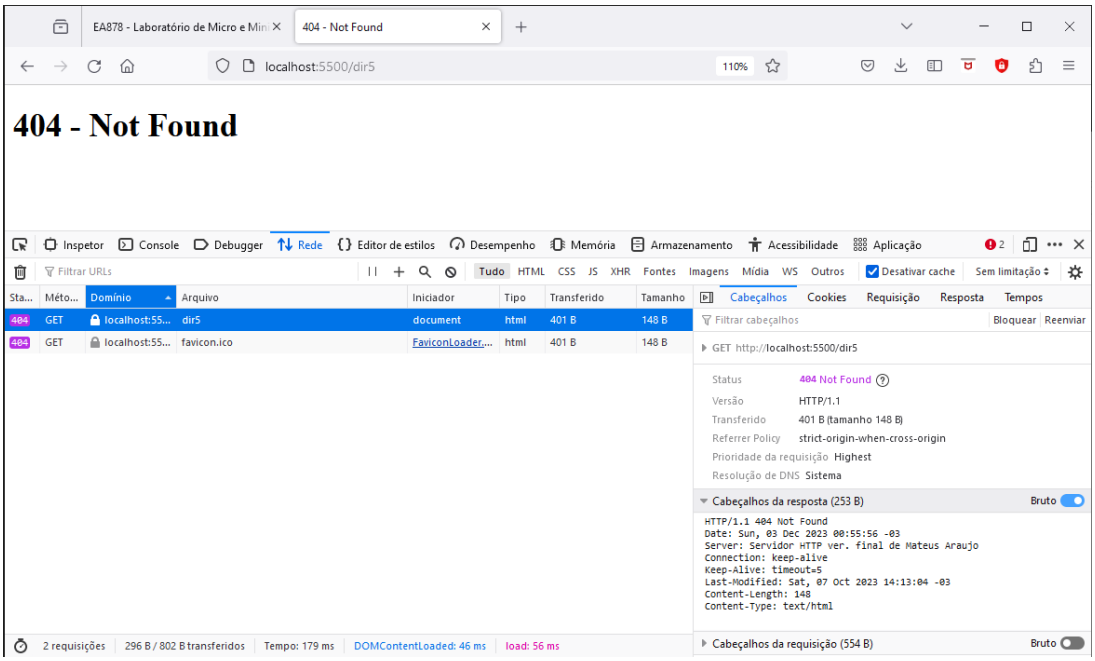


Figura 15: *resultado do teste 9. Nele podemos ver que é retornado o código de erro 404 Not Found, assim como esperado.*

10. Requisição de um diretório sem permissão de execução:

Para realizar esse caso de teste, vamos solicitar pelo browser o recurso “dir”. Podemos fazer isto ao clicarmos no link “Diretório sem permissão de execução” na seção “Casos especiais” da página principal do *web-space* de teste. O resultado obtido é apresentado na figura 16.

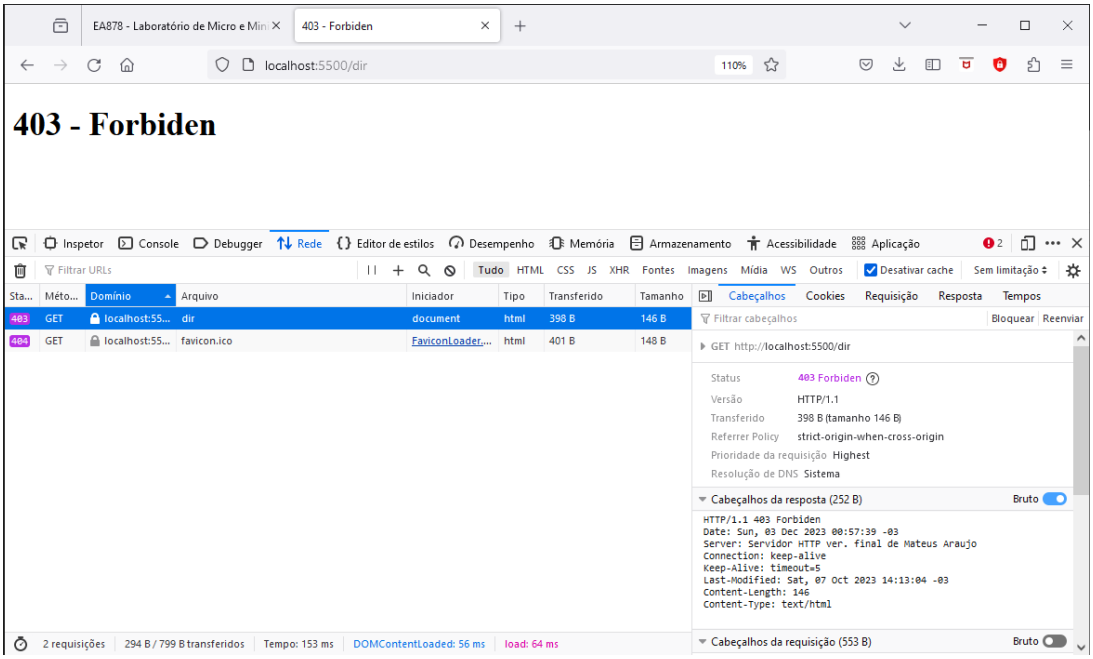


Figura 16: *resultado do teste 10. Nele podemos ver que é retornado o código de erro 403 Forbiden, assim como esperado.*

11. Requisição de arquivos de tipos jpeg, gif, tif, png e pdf:

Como nos testes anteriores já foram requisitados recursos do tipo *txt* e *html*, ainda falta testar o envio de arquivos do tipo *jpeg*, *gif*, *tiff*, *png* e *pdf* . Para realizar esses casos de teste, vamos solicitar pelo browser o recurso “unicamp.gif”, “unicamp.png”,

"unicamp.jpg", "unicamp.tif" e "teste.pdf". Podemos fazer isto ao clicarmos nos links "Formato gif", "Formato png", "Formato jpg", "Formato tif" e "Formato pdf", respectivamente, dispostos tanto na seção "Logotipos da Unicamp e da FEEC" quanto na seção "Arquivos" da página principal do *web-space* de teste. O resultado obtido é apresentado nas figuras 17 a 21.

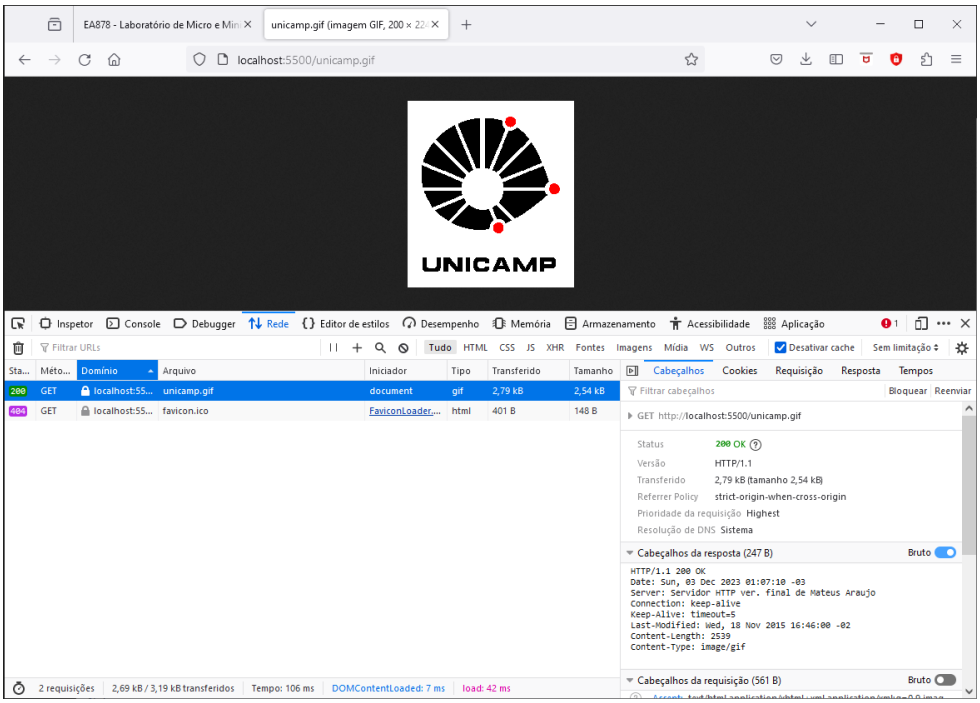


Figura 17: *resultado do teste 11 com a requisição do recurso "unicamp.gif". Nele podemos ver que o arquivo desejado foi enviado adequadamente.*

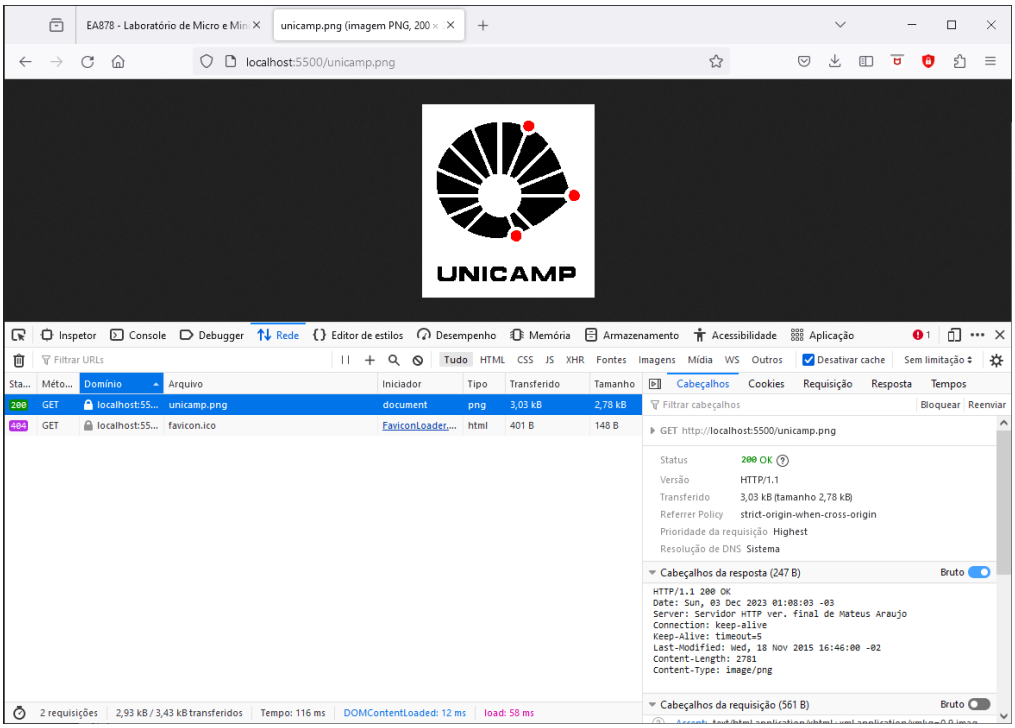


Figura 18: *resultado do teste 11 com a requisição do recurso "unicamp.png". Nele podemos ver que o arquivo desejado foi enviado adequadamente.*

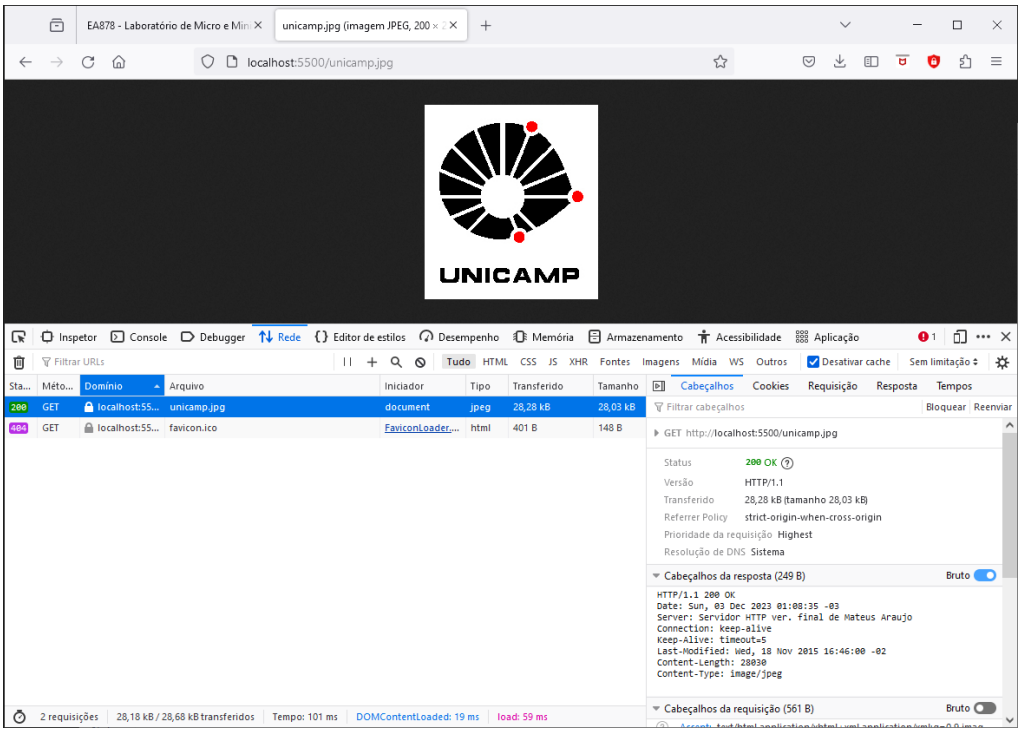


Figura 19: *resultado do teste 11 com a requisição do recurso "unicamp.jpg". Nele podemos ver que o arquivo desejado foi enviado adequadamente.*

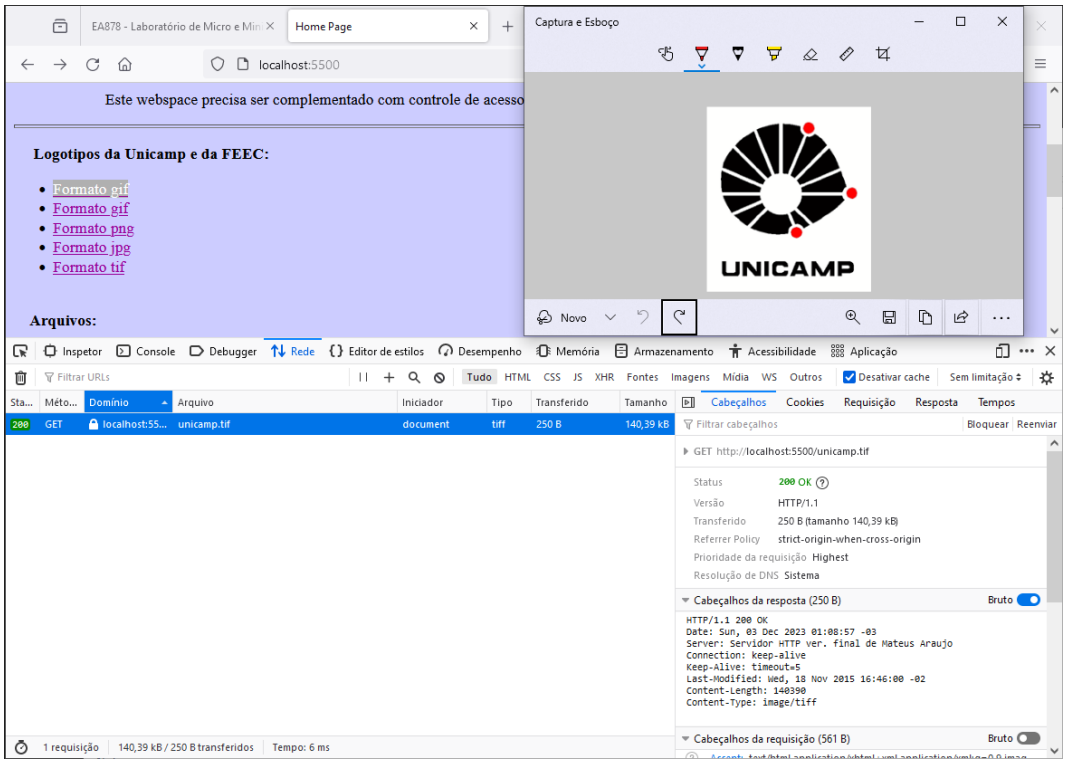


Figura 20: *resultado do teste 11 com a requisição do recurso "unicamp.tif". Nele podemos ver que o arquivo desejado foi enviado adequadamente.*

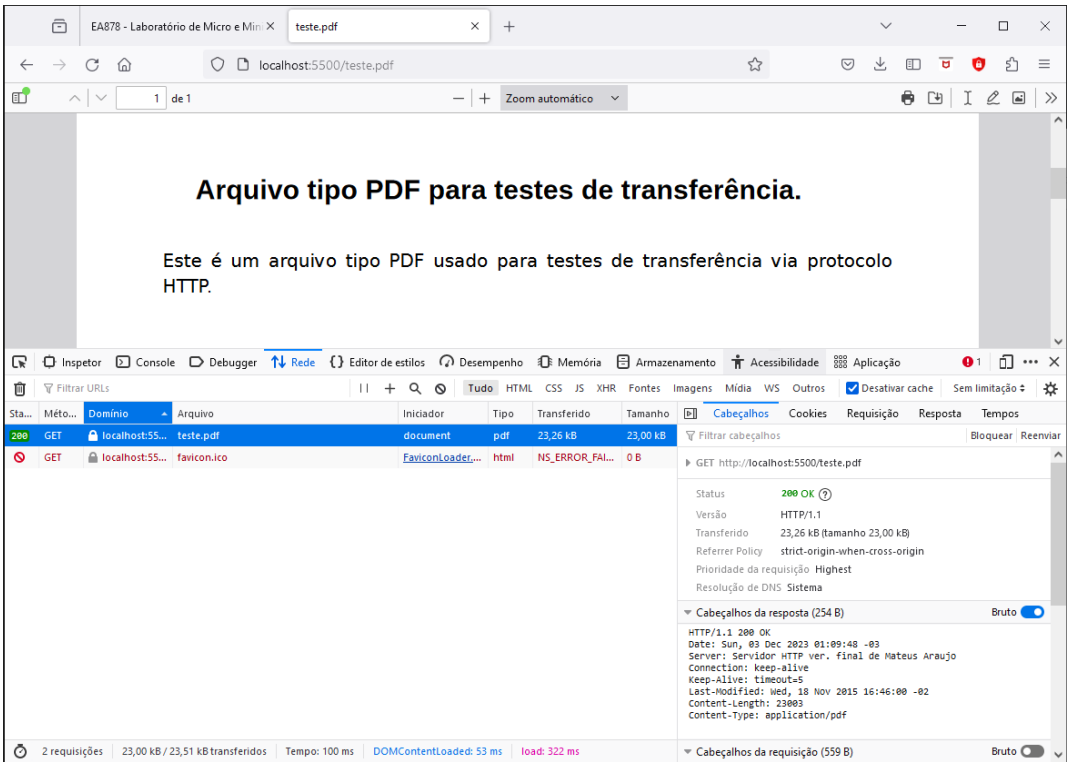


Figura 21: *resultado do teste 11 com a requisição do recurso "teste.pdf". Nele podemos ver que o arquivo desejado foi enviado adequadamente.*

12. Requisição de um recurso protegido por um arquivo .htaccess no mesmo diretório:

Para realizar esse caso de teste, vamos solicitar pelo browser o recurso "dir1/dir11". Podemos fazer isto ao clicarmos no link "Diretório com index.html e protegido por .htaccess." na seção "Casos especiais" da página principal do *web-space* de teste.

Como o arquivo é protegido por um *.htaccess*, espera-se, primeiramente que o *browser* gere uma tela para inserção de credenciais. Isso é exatamente o que ocorre, assim como apresentado na figura 22.

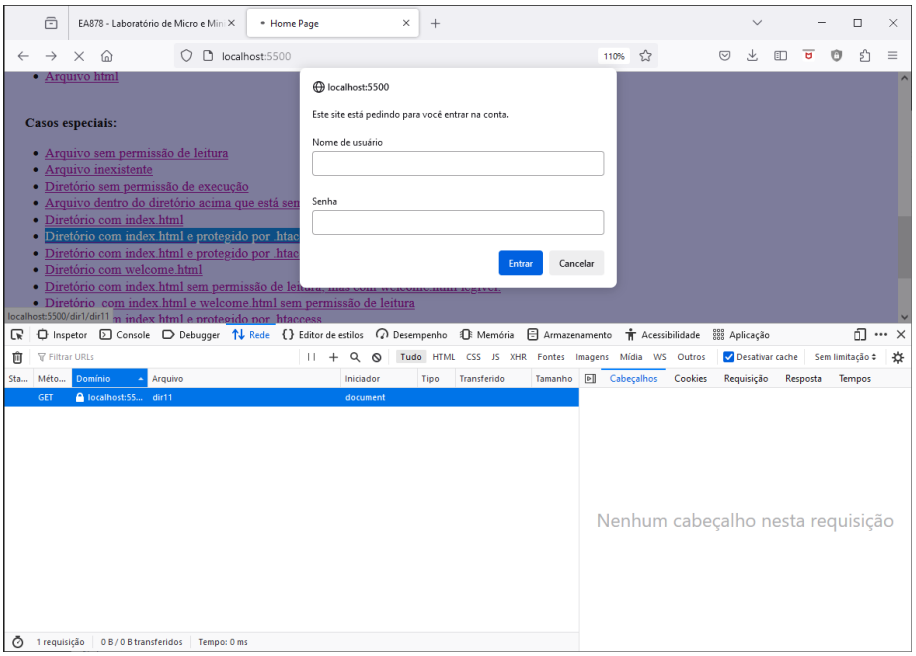


Figura 22: *primeira etapa do teste 12. A captura mostra que, ao se requisitar um recurso protegido, o browser imediatamente abres uma janela para inserção de credenciais.*

Em seguida, para testar a possibilidade da inserção de pares usuário e senha errados, podemos fornecer valores que não constam no arquivo de senhas relativo a esse diretório. Fazendo isso, obtemos simplesmente a mesma tela de inserção de credenciais novamente, que é o comportamento esperado para essa situação.

Por fim, para verificar a capacidade de retorno do recurso desejado uma vez que a autorização correta é fornecida, podemos usar a relação disposta na figura 5 e selecionar um dos conjuntos usuários e senha para o arquivo "dir11". Usando o par "user1" e "senhates" obtemos o resultado apresentado na figura 23.

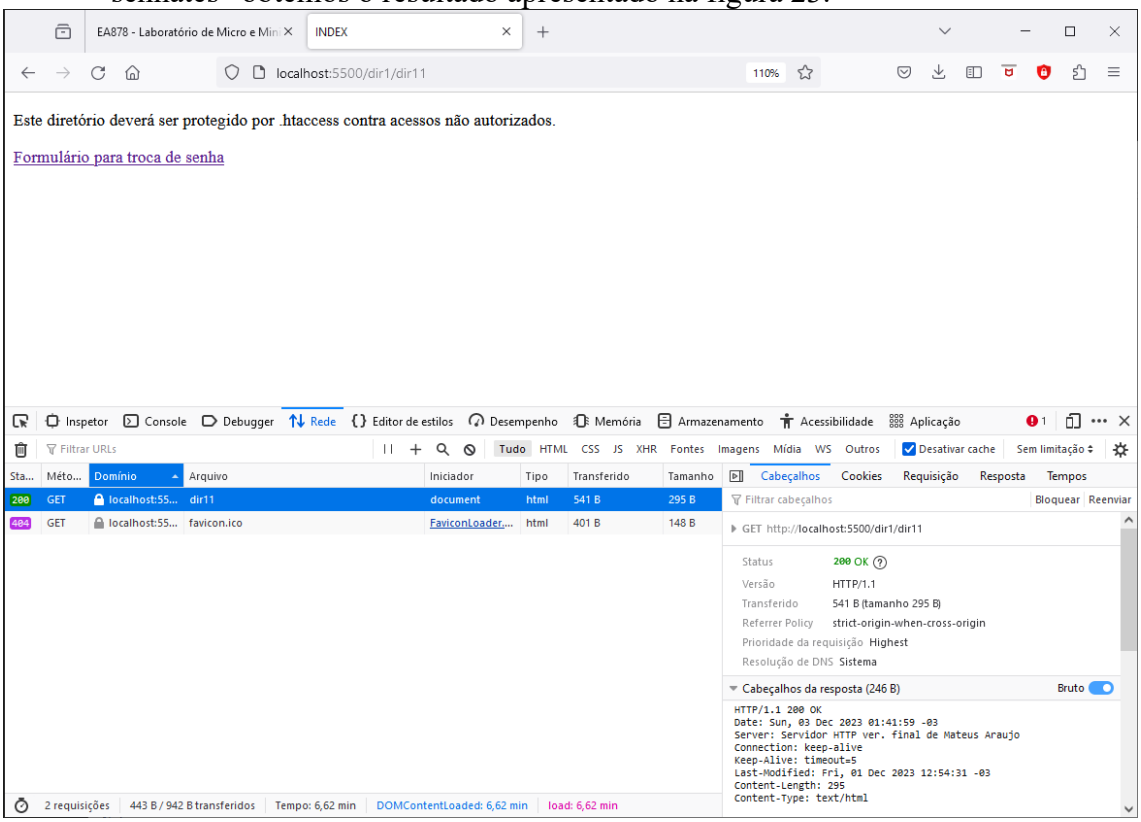


Figura 23: *segunda etapa do teste 12. A captura mostra que, ao fornecer um conjunto válido de nome de usuário e senha, o recurso buscado é retornado adequadamente.*

13. Requisição de um recurso protegido por um arquivo .htaccess no mesmo diretório, porém existindo um arquivo .htaccess no diretório acima:

Para realizar esse caso de teste, vamos solicitar pelo browser o recurso "dir1/dir11/dir111". Podemos fazer isto ao clicarmos no link "Diretório com index.html e protegido por .htaccess distinto do anterior." na seção "Casos especiais" da página principal do *web-space* de teste.

Nesse caso, espera-se que, da mesma forma que no teste anterior, o browser solicite a inserção de credenciais e somente após o fornecimento de um dos pares usuário-senha presentes no arquivo "senhas_dir111.txt" o recurso seja retornado. Assim, fornecendo o usuário "user2" e a senha "aquiaqui", o resultado para este teste foi obtido e está apresentado na figura 24.

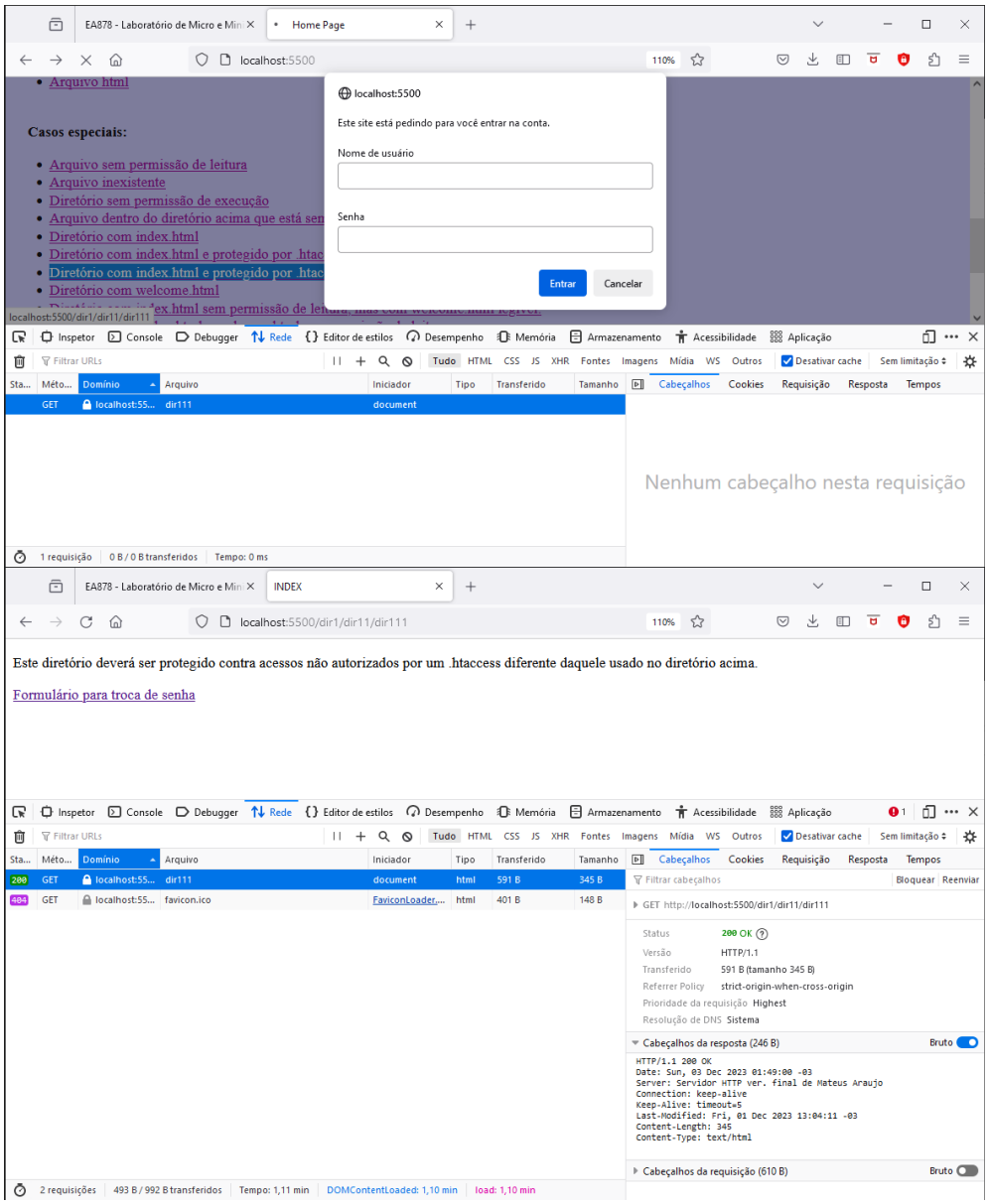


Figura 24: *resultado do teste 13. A captura superior mostra que novamente o browser solicita o fornecimento de credenciais para se acessar o recurso. A captura inferior foi obtida mediante o fornecimento de um par usuário-senha válido para o próprio diretório do recurso (dir111).*

14. Requisição de um recurso protegido por um arquivo .htaccess em um diretório acima:

Para realizar esse caso de teste, vamos solicitar pelo browser o recurso "dir4/dir41/dir411". Podemos fazer isto ao clicarmos no link "Diretório com index.html, sem .htaccess, mas protegido pelo .htaccess do nível de cima." na seção "Casos especiais" da página principal do *web-space* de teste.

Como, nesse caso, o diretório requisitado não apresenta um arquivo *.htaccess* próprio, espera-se que o arquivo deste tipo presente no diretório um nível acima faça sua proteção. Dessa forma, espera-se, novamente, que o *browser* solicite o fornecimento de credencias e que a página requisitada só seja entregue mediante o uso de um usuário e senha presente no arquivo "senhas_dir41.txt". Selecionando o conjunto "user1" e "mateus123", obtemos como resultado do teste o exposto na figura 25.

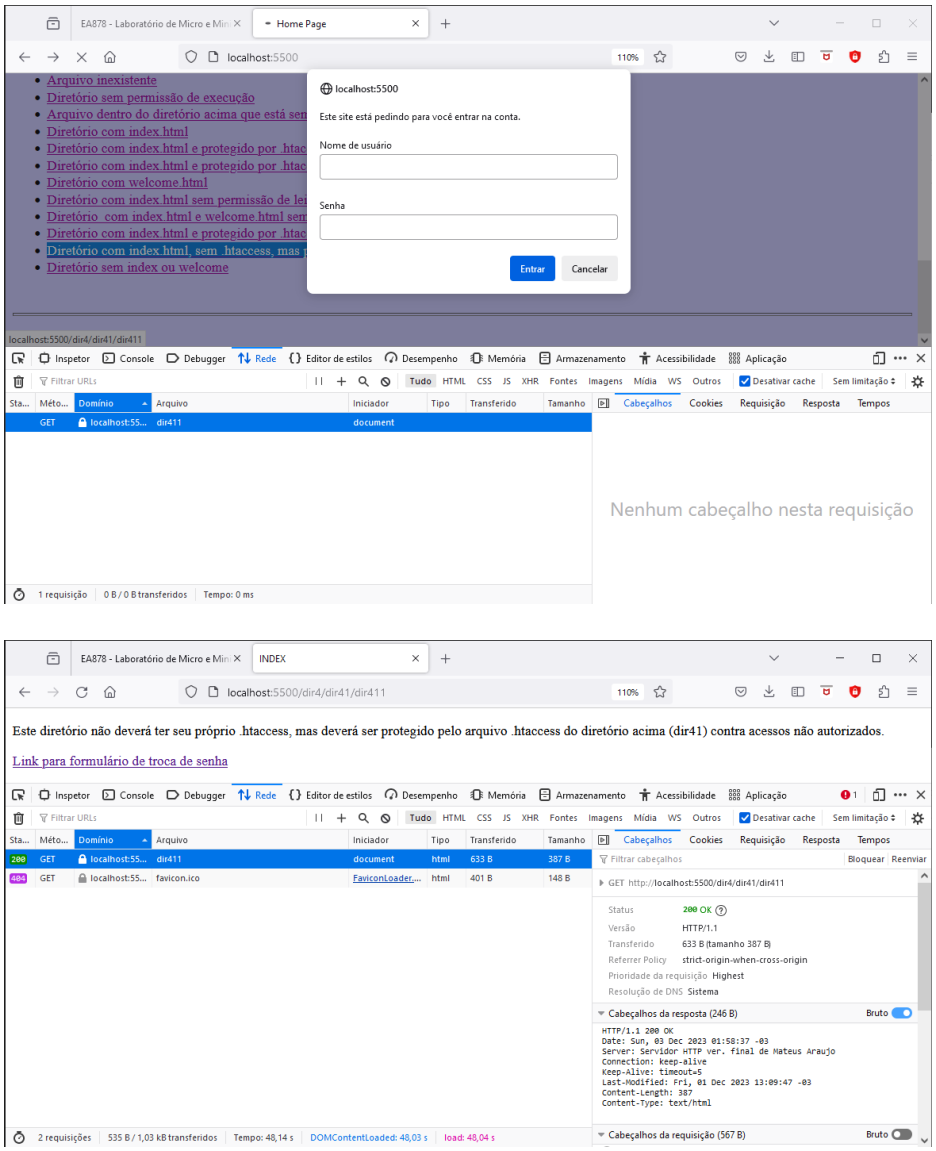


Figura 25: *resultado do teste 14. A captura superior mostra que novamente o browser solicita o fornecimento de credenciais para se acessar o recurso. A captura inferior foi obtida mediante o fornecimento de um par usuário-senha válido para o diretório acima do recurso (dir41).*

15. Envio de formulário de troca de senhas corretamente preenchido para alteração de senha de usuário:

Agora que foram concluídos os testes de acesso a recursos protegidos, vamos iniciar os testes da funcionalidade de alteração de senhas por meio de formulário. Para tal, será utilizado a página de formulário existente no caminho "dir4/dir41/dir411/form.html".

Como o diretório "dir411" não apresenta arquivo *.htaccess* próprio, mas o diretório "dir41" apresenta, espera-se que seja possível alterar a senha de algum usuário cadastrado no arquivo de senhas "senha_dir41.txt" (ao passo que é o arquivo *.htaccess* desse diretório que atua sobre o recurso em "dir411"). Dessa forma, para comprovar a eficácia do teste, vale destacar o conteúdo inicial do arquivo "senha_dir41.txt", o qual está exposto na figura 4.

Dando continuidade ao teste, selecionamos o usuário "user1" (cuja senha inicial é "mateus123") e preenchemos os campos do formulário adequadamente, escolhendo a nova senha como sendo "novasenh".

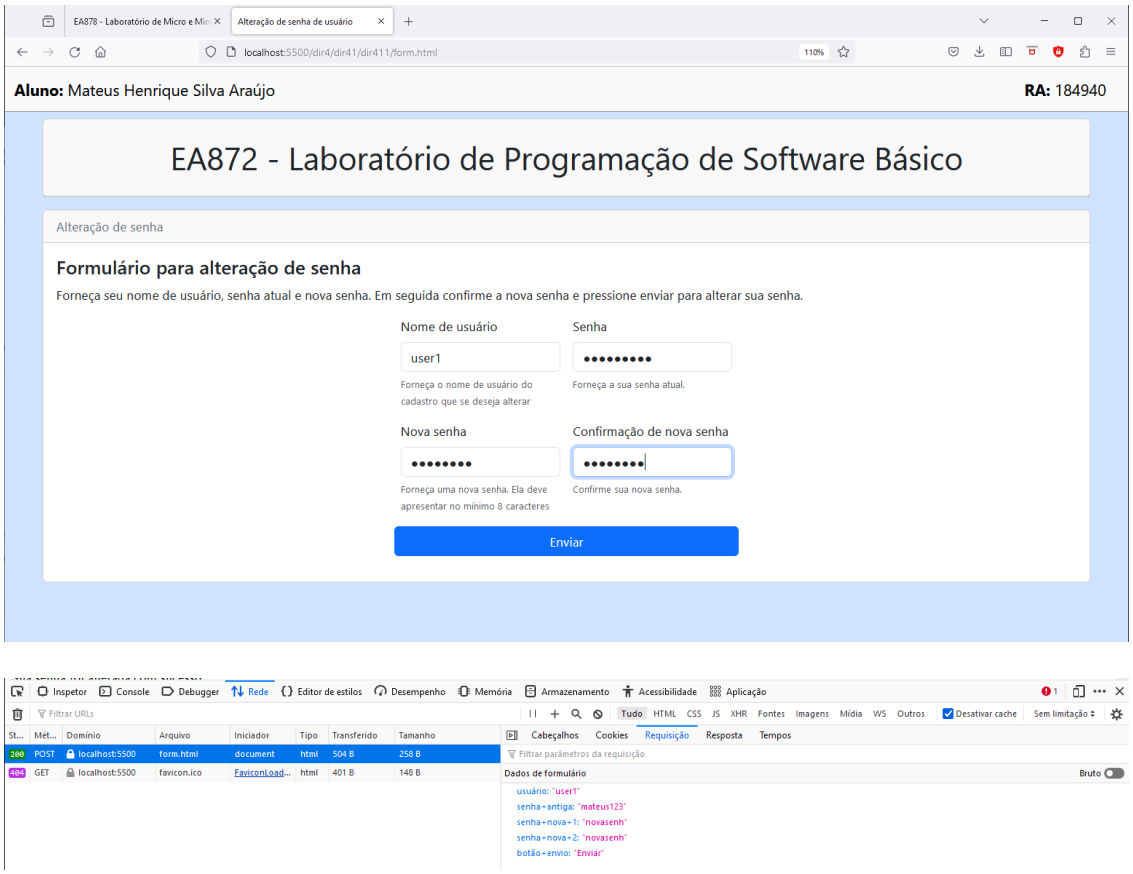


Figura 26: *primeira etapa do teste 15. A captura mostra o preenchimento do formulário de troca de senha com os valores "user1" para o campo "Nome de usuário", "mateus123" para o campo "senha", "novasenh" para o campo "Nova senha" e novamente "novasenh" para o campo "Confirmação de nova senha".*

Como os campos foram preenchidos adequadamente, isto é, o nome de usuário e a senha fornecida constam no arquivo de senhas indicado pelo *.htaccess* que protege o recurso, bem como a nova senha e sua confirmação são idênticas, espera-se que o servidor faça a alteração da senha cadastrada no arquivo e envie uma página indicando sucesso da operação. Fazendo o envio do formulário acima, obtemos exatamente a página esperada, assim como mostra a figura 27.

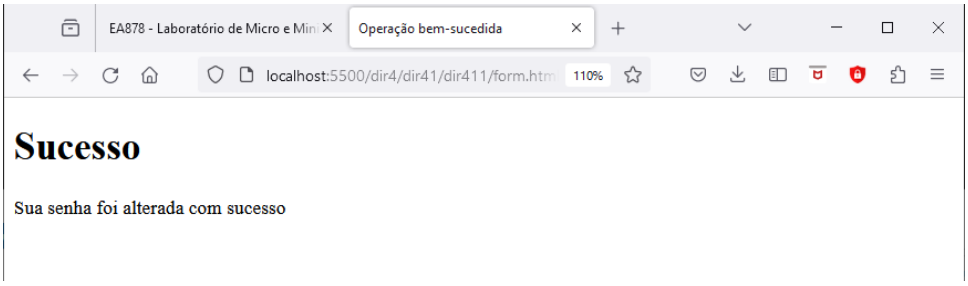


Figura 27: *segunda etapa do teste 15. A captura mostra que o servidor retornou uma página html indicando o sucesso da troca de senha.*

Para confirmar a efetuação da troca de senhas, podemos verificar novamente o conteúdo do arquivo "senhas_dir41.txt" e comparar o valor da senha do usuário "user1" com a saída da função `crypt()` usando *salt* "EA" e parâmetro de entrada "novasenh". A imagem para comparação é a apresentada na figura 28.

```
mateus@DESKTOP-KSDIOPD:~/EA872/Lab13$ ./cripto novasenh EA

  ( Salt = EA ) + ( Password = novasenh ) ==> ( crypt = EAdjbXXxtvvXM )

mateus@DESKTOP-KSDIOPD:~/EA872/Lab13$ cat senhas_dir41.txt -
user2:EA1AkHKq4eyVY
user5:EABWmnk3z9qEE
user1:EAdjbXXxtvvXM
user3:EAtarOXmxIxTk
user4:EAWPYXYHdCiXA

mateus@DESKTOP-KSDIOPD:~/EA872/Lab13$
```

Figura 28: *comparação da saída da função crypt() usando-se a nova senha definida para o usuário cuja senha foi alterada e o conteúdo do arquivo de senhas afetado pela alteração. Conclui-se que a alteração foi realmente efetuada*

Outro detalhe que vale mencionar é que, como o formulário de troca de senhas foi acessado com as credenciais do usuário “user1”,caso o mesmo usuário tente voltar para a página com o formulário de alteração de senhas ou acessar outro arquivo do diretório “dir411”, será necessário repetir o procedimento de autenticação, fornecendo, dessa vez, sua nova senha recém alterada.

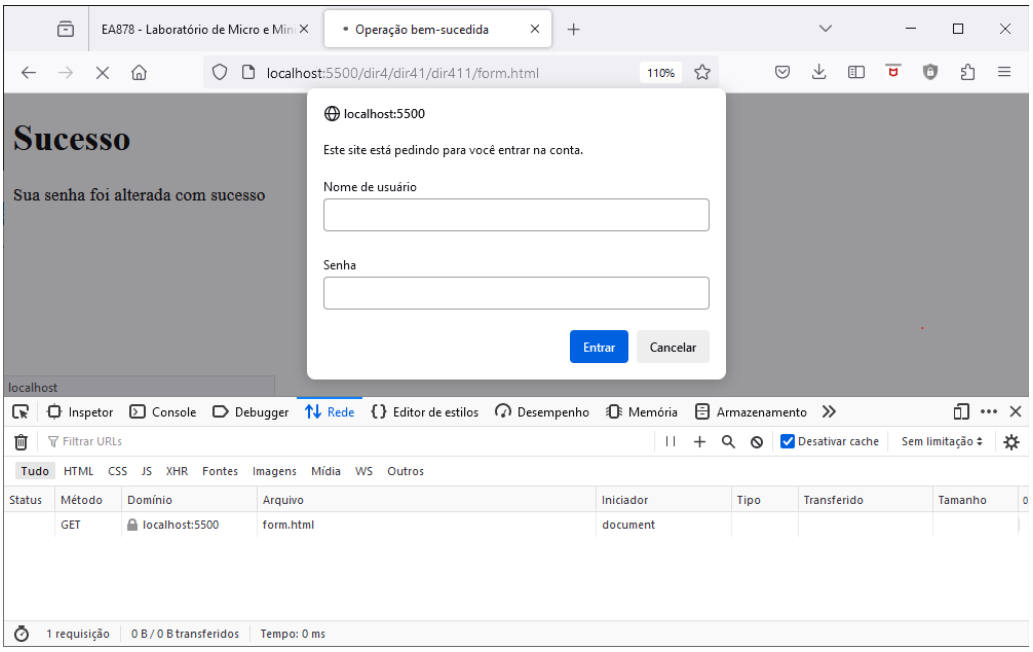
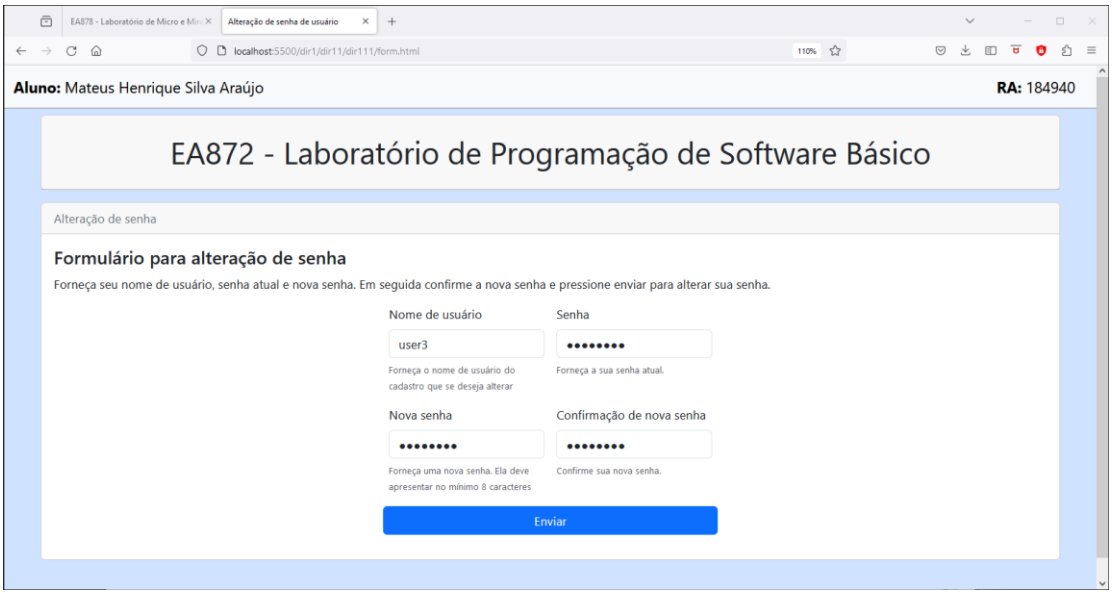


Figura 29: *última etapa do teste 15. A captura comprova que o servidor realmente alterou o conteúdo do arquivo “senhas_dir41.txt”, pois faz-se necessário a autenticação novamente do usuário “user1”, agora com sua nova senha.*

Para finalizar esse caso de teste, vamos realizar procedimento análogo usando como alvo o recurso “dir1/dir11/dir111”, o qual apresenta um arquivo de senha distinto do testado anteriormente. Dessa forma, lembrando-se que o arquivo de senhas aletrado dessa ver será “senhas_dir111.txt” (cujo conteúdo inicial também está exposto na figura 4) as capturas das etapas do teste estão expostas nas figuras 30, 31 e 32.



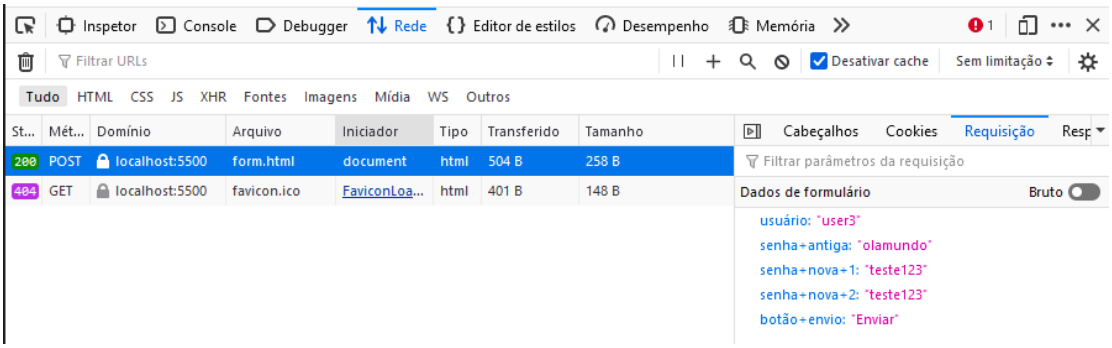


Figura 30: *A captura mostra o preenchimento do formulário de troca de senha com os valores "user3" para o campo "Nome de usuário", "olamundo" para o campo "senha", "teste123" para o campo "Nova senha" e novamente "teste123" para o campo "Confirmação de nova senha". Destaca-se que agora a operação é sobre o recurso "dir1/dir11/dir111".*

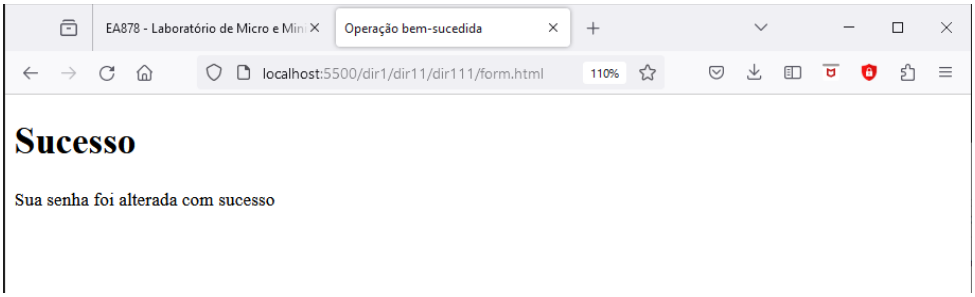


Figura 31: *A captura mostra que o servidor retornou uma página html indicando o sucesso da troca de senha. Destaca-se que agora a operação é sobre o recurso "dir1/dir11/dir111".*

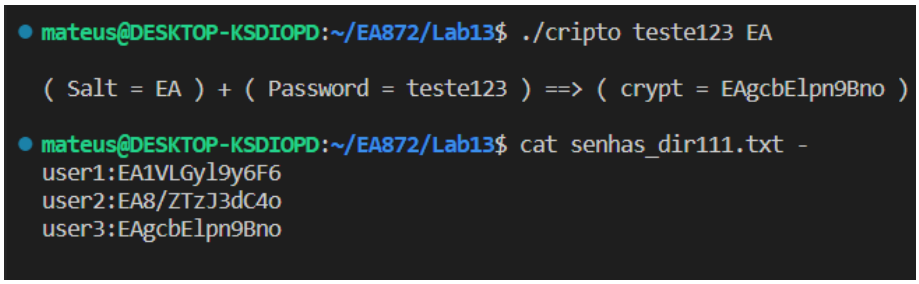


Figura 32: *comparação da saída da função crypt() usando-se a nova senha definida para o usuário cuja senha foi alterada e o conteúdo do arquivo de senhas afetado pela alteração. Conclui-se que a alteração foi realmente efetuada.*

Finalizado este teste, os arquivos de senhas foram retornados para suas formas padrão com o intuito de manter-se a coerência com o mapeamento apresentado na figura 4.

16. Envio de formulário de troca de senhas com usuário que não consta no arquivo de senhas:

Como demostrado no caso de teste anterior, o aplicativo servidor é capaz de efetivamente realizar a troca de senha de um usuário por meio do formulário de troca de senhas. Cabe, agora, testar os eventuais erros que podem ocorrer nesse processo.

Primeiramente, vamos usar do recurso “dir1/dir11/form.html” para tentar alterar a senha de um usuário que não consta no arquivo “senhas_dir11.txt". Para tal, será inserido no campo “Nome de usuário” o valor “usuarioFalso” e valores aleatórios nos demais campos. Vale destacar que, para que o teste funcione, os valores dos campos “Nova senha” e “Confirmação de nova senha” devem condizer. A figura 33 apresenta o preenchimento do formulário em questão com os valores indicados.

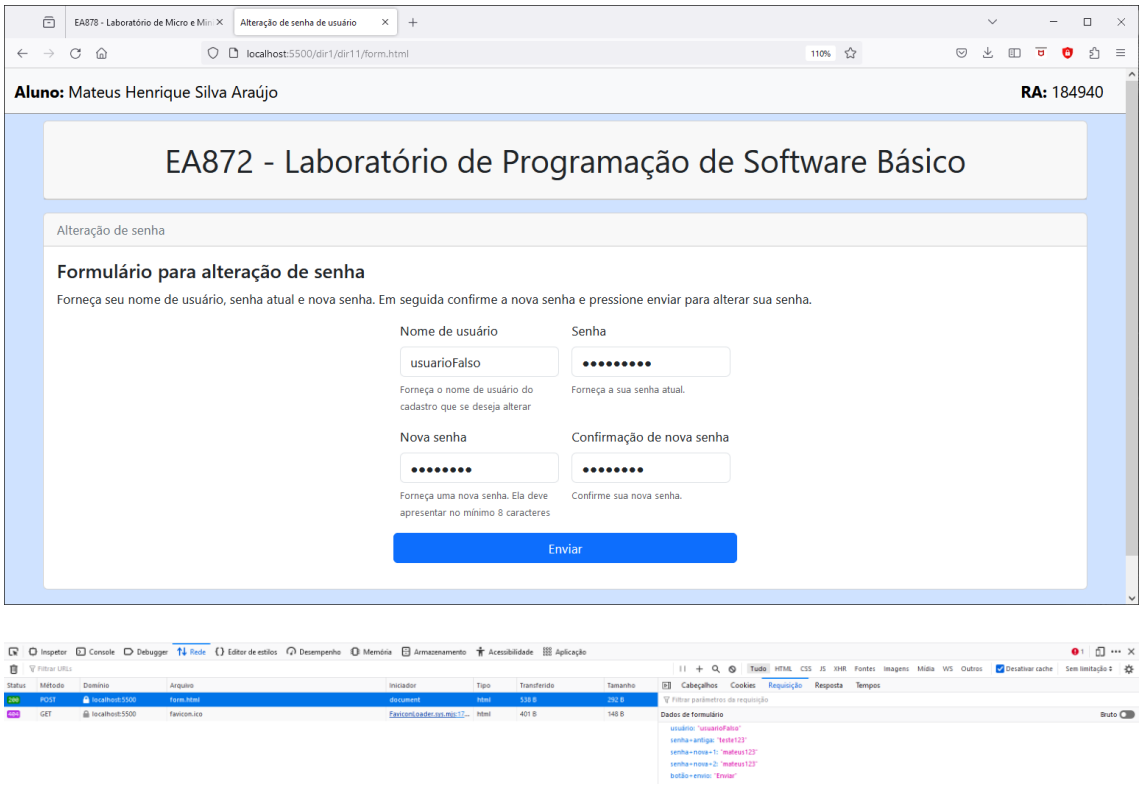


Figura 32: *preenchimento do formulário “dir1/dir11/form.html” com um usuário que não consta no arquivo de senhas “senhas_dir11.txt”, assim como é possível ver na figura 4.*

Ao enviar o formulário preenchido, o resultado obtido, que está apresentado na figura 33, é uma página *html* indicando que ou o usuário ou a senha estão incorretos. Esta é justamente a resposta esperada nesse tipo de situação. Além disso, podemos confirmar que o conteúdo do arquivo “senhas_dir11.txt” permanece o mesmo ao compararmos a figura 34 (obtida após este teste) e a figura 4.

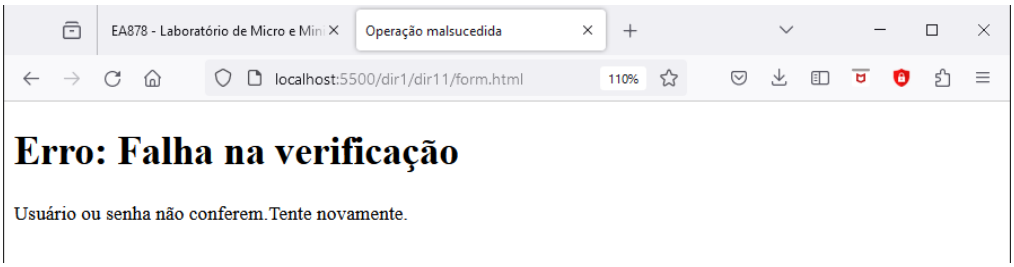


Figura 33: *página de resposta enviada pelo servidor após submissão do formulário preenchido com usuário que não consta no arquivo de senhas.*

```
mateus@DESKTOP-KSDIOPD:~/EA872/Lab13$ cat senhas_dir11.txt -
user5:EAh11UcXydMIM
user3:EA/pH0jmlKfP6
user4:EAn8Hfdrbbtvg
user1:EACxq6nE/7pmw
user2:EAuCql5t/bxq2
```

Figura 34: *conteúdo do arquivo "senhas_dir11.txt" após a realização das etapas anteriores do teste 16. Comparando-o com o conteúdo inicial, apresentado na figura 4, vemos que ele se mantém o mesmo.*

17. Envio de formulário de troca de senhas com senha antiga que não condiz com a senha cadastrada do usuário:

Outra possibilidade de erro que deve ser testada é a submissão de um formulário contendo o campo "Nome de usuário" com um valor que consta no arquivo de senhas, mas com uma senha distinta. Para tal, usaremos do mesmo recurso do teste anterior, porém, dessa vez, preenchendo o campo "Nome de usuário" com o valor "user1" (que é um usuário válido no arquivo "senhas_dir11.txt") e o campo "Senha" com o valor "aaabbbcc" (que é uma senha válida no arquivo, porém pertencente a "user3"). Para que esse teste seja efetivo, os campos de "Nova senha" e "Confirmação de nova senha" devem ser preenchidos adequadamente, pois o servidor verifica primeiro se esses valores são iguais para depois processar o nome de usuário. A figura 34 apresenta o preenchimento do formulário em questão com os valores indicados.

Ao enviar o formulário preenchido, o resultado obtido, que está apresentado na figura 35, é a mesma página *html* recebida no teste anterior. Novamente o comportamento do servidor é condizente com o esperado. Além disso, podemos confirmar, mais uma vez, que o conteúdo do arquivo "senhas_dir11.txt" se mantém inalterado, assim como expõe a figura 36.

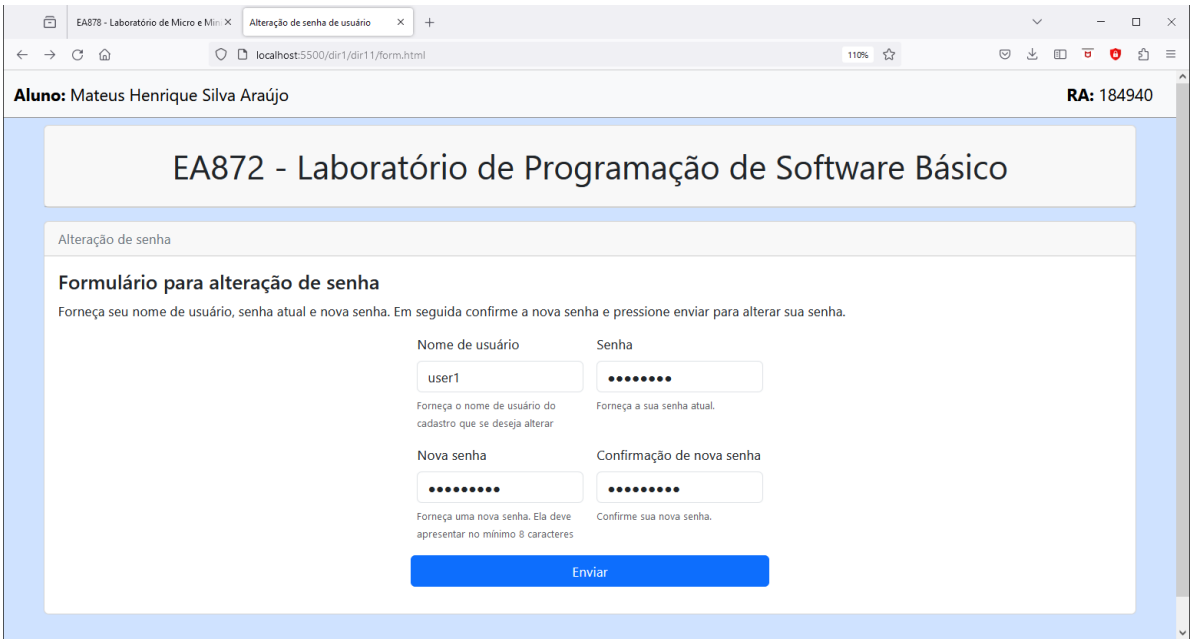


Figura 34: *preenchimento do formulário "dir1/dir11/form.html" com um usuário que consta no arquivo de senhas "senhas_dir11.txt", porém com uma senha que não corresponde a sua senha cadastrada.*

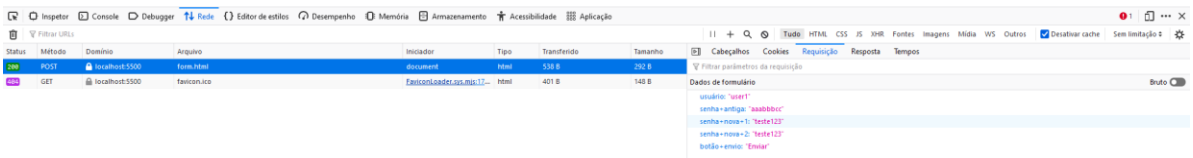


Figura 35: : *página de resposta enviada pelo servidor após submissão do formulário preenchido com sennha que não condiz com a senha atual de um usuário cadastrado no arquivo de senhas.*

```
mateus@DESKTOP-KSDIOPD:~/EA872/Lab13$ cat senhas_dir11.txt -
user5:EAh11UcXydMIM
user3:EA/pH0jmlKfP6
user4:EAn8Hfdrbbtvg
user1:EACxq6nE/7pmw
user2:EAuCql5t/bxq2
```

Figura 36: *conteúdo do arquivo "senhas_dir11.txt" após a realização das etapas anteriores do teste 17. Comparando-o com o conteúdo inicial, apresentado na figura 4, vemos que ele se mantém o mesmo.*

18. Envio de formulário de troca de senhas com nova senha e confirmação de nova senha não condizentes:

Por fim, um último teste quanto à funcionalidade de alteração de senha por meio do que vale ser feito é a submissão de um formulário com valores dos campos "Nome de usuário" e "Senha" que identificam um cadastro válido no arquivo de senhas, porém com os campos "Nova senha" e "Confirmação de nova senha" distintos. Para tal, usaremos do mesmo recurso do teste anterior, desta vez, preenchendo o campo "Nome de usuário" com o valor "user1" (que é um usuário válido no arquivo "senhas_dir11.txt") , o campo "Senha" com o valor "senhates" (que é a senha cadastrada para esse usuário em tal arquivo) e os campos "Nova senha" e "Confirmação de nova senha" com valores distintos quaisquer. . A figura 37 apresenta o preenchimento do formulário em questão com os valores indicados.

Ao enviar o formulário preenchido, o resultado obtido, que está apresentado na figura 38, é uma página *html* indicando que a troca de senha não foi efetuada, pois a nova senha e sua confirmação não condizem. Esta é a resposta esperada nesse tipo de situação. Além disso, podemos confirmar, mais uma vez, que o conteúdo do arquivo "senhas_dir11.txt" se mantém inalterado, assim como expõe a figura 39.

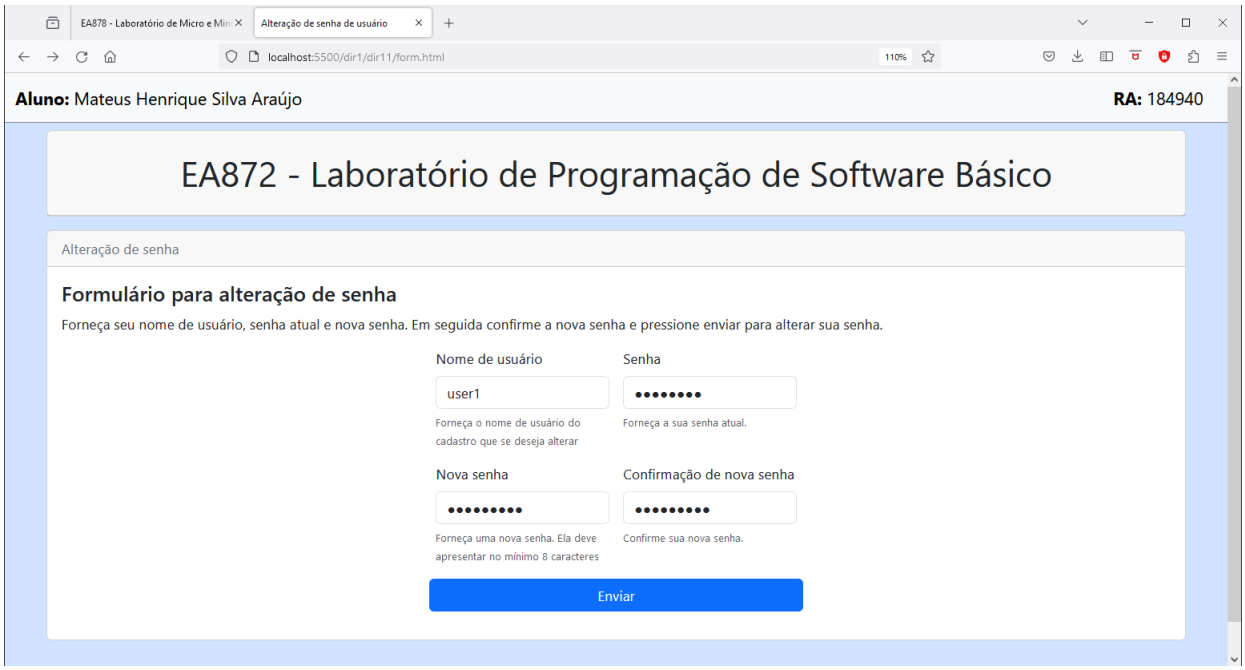


Figura 37: *preenchimento do formulário "dir1/dir11/form.html" com um usuário que consta no arquivo de senhas "senhas_dir11.txt", e com sua senha cadastrada, porém com nova senha e confirmação distintas.*

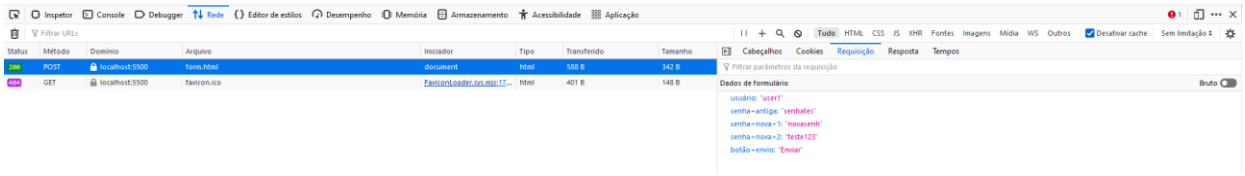


Figura 38: *página de resposta enviada pelo servidor após submissão do formulário preenchido com informações de teste.*

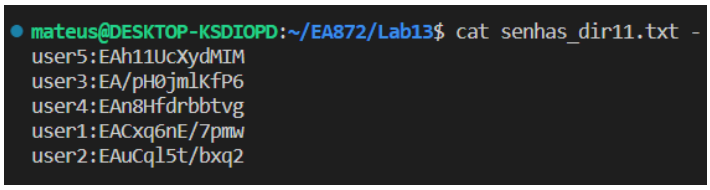


Figura 39: *conteúdo do arquivo "senhas_dir11.txt" após a realização das etapas anteriores do teste 18. Comparando-o com o conteúdo inicial, apresentado na figura 4, vemos que ele se mantém o mesmo.*

19. Requisição com o método OPTIONS:

Para testar a capacidade de resposta do servidor a requisições com o método OPTIONS, podemos utilizar do programa *telnet*, ao passo que os *browsers* comuns não geram esse tipo de requisição. Dessa forma, a figura 40 apresenta a conexão com o servidor, o envio da requisição e a resposta do servidor para esse caso.

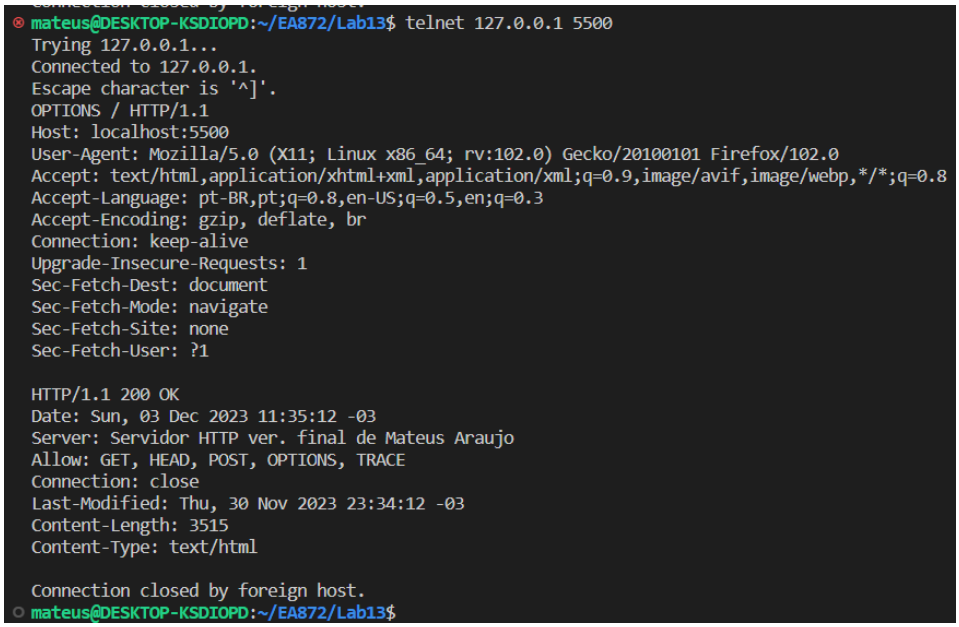


Figura 40: *conexão com o servidor por meio do programa telnet, envio de requisição com método OPTIONS e resposta recebida em retorno. Podemos ver que, assim como esperado, o servidor envia uma mensagem contendo o cabeçalho "Allow", informando quais são os métodos HTTP que ele suporta processar.*

20. Requisição com o método TRACE:

Para testar a capacidade de resposta do servidor a requisições com o método TRACE, podemos utilizar do programa *telnet*, ao passo que os *browsers* comuns não geram esse tipo de requisição. Dessa forma, a figura 41 apresenta a conexão com o servidor por meio, o envio da requisição com método e a resposta do servidor para esse teste.

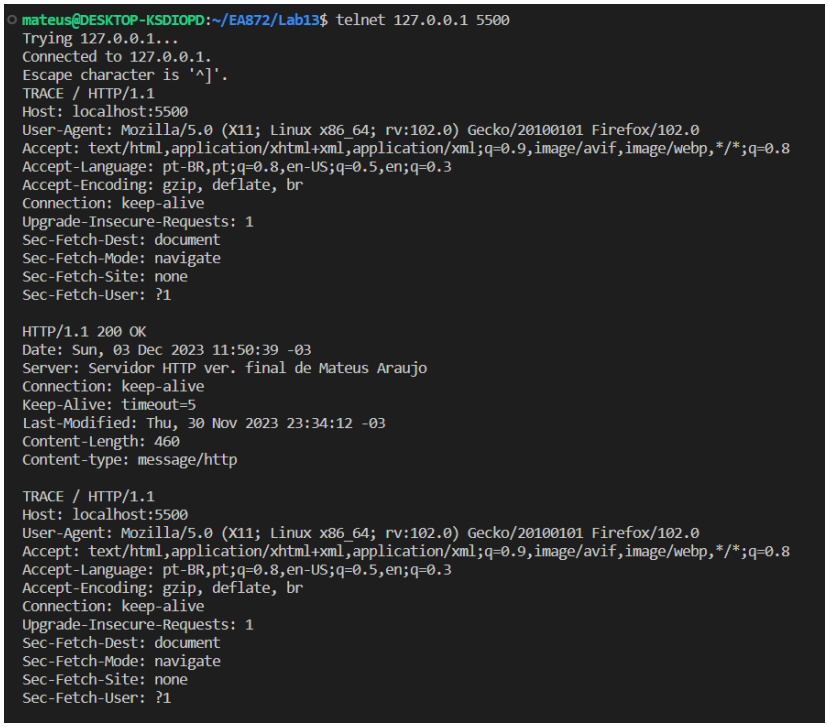


Figura 41: *conexão com o servidor por meio do programa telnet, envio de requisição com método TRACE e resposta recebida em retorno. Podemos ver que, assim como esperado, o servidor envia uma mensagem contendo em seu corpo a exata mesma mensagem que ele recebeu como requisição.*

21. Requisições para recursos fora do *web-space*:

Para testar o confinamento ao *web-space* estabelecido pelo servidor, vamos usar do aplicativo *telnet*, ao passo que os *browsers* comuns não inserção de diretórios ".." no caminho do recurso, para solicitar o conteúdo do arquivo "mapeamento.csv" (apresentado na figura 4) que se localiza no diretório acima do *web-space* de teste. Assim,

a figura 42 apresenta o procedimento de teste e a resposta do servidor, validando que o servidor impede o acesso de recursos externos ao espaço roteado.

```
mateus@DESKTOP-KSDIOPD:~/EA872/Lab13$ telnet 127.0.0.1 5500
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
GET /../mapeamento.csv HTTP/1.1
Host: localhost:5500
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: pt-BR,pt;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
Sec-Fetch-User: ?1

HTTP/1.1 403 Forbiden
Date: Sun, 03 Dec 2023 14:44:00 -03
Server: Servidor HTTP ver. final de Mateus Araujo
Connection: keep-alive
Keep-Alive: timeout=5
Last-Modified: Sat, 07 Oct 2023 14:13:04 -03
Content-Length: 146
Content-Type: text/html

<!DOCTYPE html>
<html>
    <head>
        <title>403 - Forbiden</title>
    </head>
    <body>
        <h1>403 - Forbiden</h1>
    </body>
</html>
```

Figura 42: *conexão com o servidor por meio do programa telnet, envio de requisição com método GET requisitando um recurso externo ao servidor e resposta recebida em retorno. Podemos ver que, assim como esperado, o servidor envia o erro 403 Forbiden, pois o acesso ao exterior do web-space é proibido.*

22. Tratamento de conexões que demoram muito para enviar requisições:

Para testar a capacidade do servidor de enviar uma resposta do tipo “Request Timeout” quando um cliente estabelece uma conexão, porém demora muito (no caso dessa implementação, mais que 5 segundos) para escrever algo no soquete, podemos usar o aplicativo *telnet* para se conectar ao servidor e aguardar a ocorrência do erro. A figura 43 apresenta a resposta enviada pelo servidor quando aproximadamente 5 segundos se passam após a chamada do programa *telnet*.

```
mateus@DESKTOP-KSDIOPD:~/EA872/Lab13$ telnet 127.0.0.1 5500
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
HTTP/1.1 408 Request Timeout
Date: Sun, 03 Dec 2023 14:51:47 -03
Server: Servidor HTTP ver. final de Mateus Araujo
Connection: close
Last-Modified: Tue, 21 Nov 2023 14:30:06 -03
Content-Length: 276
Content-Type: text/html

<!DOCTYPE html>
<html>
    <head>
        <title>408 - Request Timeout</title>
    </head>
    <body>
        <h1>408 - Request Timeout</h1>
        <p> The server closed connection because it did not receive a complete request message within 5 seconds</p>
    </body>
</html>Connection closed by foreign host.
```

Figura 43: *resposta do servidor para o caso no qual o cliente realiza uma conexão e demora muito para escrever algo no soquete. Esse tipo de situação também ocorre após conexões do tipo “keep-alive” realizada por browsers nas quais o browser já terminou de escrever todas suas requisições*

23. Requisição quando o número máximo de threads em operação é atingido:

Para testar a capacidade do servidor de enviar uma resposta do tipo “Service Unavailable” quando o máximo de threads ativas foi atingido e o servidor recebe mais um pedido de conexão, podemos reduzir o número máximo de thread (parâmetro fornecido pela linha de comando) para 1 e realizarmos 2 requisições simultâneas. A figura 44 apresenta a resposta enviada pelo servidor para um dos browsers. Nela, nota-se que a página com o erro correspondente foi enviada conforme o esperado.

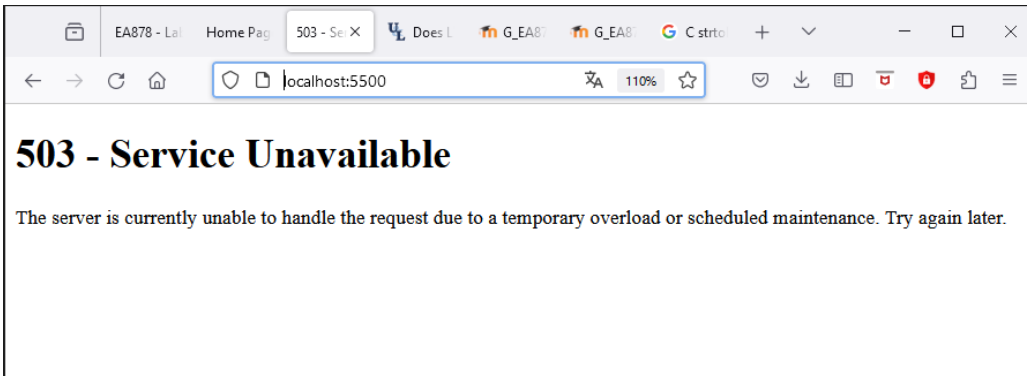


Figura 44: *página de resposta enviada pelo servidor quando uma nova requisição de conexão é feita enquanto o número de threads ativas é igual ao máximo estabelecido*

24. Testes de vazamento de memória:

Durante a maioria dos testes acima, o programa servidor estava sendo executado em conjunto com o programa *valgrind* para se buscar possíveis vazamentos de memória durante sua operação. Nesse sentido, assim como é possível ver na figura 45, mesmo após repetidos testes com sequências de operações distintas, toda a memória alocada pelo programa foi devidamente liberada. Isto indica que o programa apresenta robustez contra vazamento de memória, que é uma qualidade desejada para uma aplicação útil.

```
mateus@DESKTOP-KSDIOPD:~/EA872/Lab13/servidor_multithread$ valgrind -s --leak-check=full --show-leak-kinds=all --track-origins=yes ./servidor /home
/mateus/EA872/Lab13/webspace_para_testes 5500 20 UTF-8
==19559== Memcheck, a memory error detector
==19559== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==19559== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==19559== Command: ./servidor /home/mateus/EA872/Lab13/webspace_para_testes 5500 20 UTF-8
==19559==
Avisos!
        - A codificação dos arquivos html/txt do servidor está sendo definida para UTF-8;
        - O número máximo de threads do servidor está sendo configurado para 20;
        - A porta de operação do servidor está sendo definida para 5500;
>> Server ready to recieve connections!
^C
Termination signal recieved! Server closed...
==19559==
==19559== HEAP SUMMARY:
==19559==     in use at exit: 0 bytes in 0 blocks
==19559==   total heap usage: 7,054 allocs, 7,054 frees, 744,903 bytes allocated
==19559==
==19559== All heap blocks were freed -- no leaks are possible
==19559==
==19559== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
mateus@DESKTOP-KSDIOPD:~/EA872/Lab13/servidor_multithread$ valgrind -s --leak-check=full --show-leak-kinds=all --track-origins=yes ./servidor /home
/mateus/EA872/Lab13/webspace_para_testes 5500 20 UTF-8
==16707== Memcheck, a memory error detector
==16707== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==16707== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==16707== Command: ./servidor /home/mateus/EA872/Lab13/webspace_para_testes 5500 20 UTF-8
==16707==
Avisos!
        - A codificação dos arquivos html/txt do servidor está sendo definida para UTF-8;
        - O número máximo de threads do servidor está sendo configurado para 20;
        - A porta de operação do servidor está sendo definida para 5500;
>> Server ready to recieve connections!
^C
Termination signal recieved! Server closed...
==16707==
==16707== HEAP SUMMARY:
==16707==     in use at exit: 0 bytes in 0 blocks
==16707==   total heap usage: 12,305 allocs, 12,305 frees, 1,736,882 bytes allocated
==16707==
==16707== All heap blocks were freed -- no leaks are possible
==16707==
==16707== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Figura 45: *resultados da execução do programa servidor em conjunto com o programa valgring. Vale destacar que várias sequências de testes foram realizadas em cada execução apresentada, o que explicita a robustez contra vazamento de memória da aplicação.*

4) Conclusões:

4.1) Limitações da implementação desenvolvida:

Assim como já mencionado anteriormente, o aplicativo desenvolvido apresenta todas as características especificadas como essenciais pelo roteiro da atividade. Porém, ainda existe uma distância considerável entre ele e um servidor HTTP profissional. Dentre as diferenças que causam esse distanciamento, ignorando questões de eficiência e velocidade, está a incapacidade de se processar qualquer tipo de requisição POST, não somente aquelas que contém uma resposta padronizada com campos de formulários

específicos. Outro detalhe é que o aplicativo desenvolvido também não apresenta um comportamento adequado para requisições de métodos usuais como PUT e DELETE.

Além disso, ao se analisar a qualidade do código desenvolvido e a aplicação das boas práticas de programação, nota-se que a implementação atual é muito restritiva à extensão, ou seja, são necessárias muitas modificações em diversas funções diferentes para que uma nova funcionalidade possa ser inserida. Este ponto foi um viés considerado desde as primeiras versões da aplicação e houve um esforço considerável para se evita-lo o máximo possível. Entretanto, dado ao pouco conhecimento pessoal sobre práticas aplicáveis na programação estruturada que evitam tal característica, o produto resultante acabou ficando mais engessado que o pretendido.

Por fim, ressalta-se que, desde a versão na qual o procedimento de autorização foi primeiramente disponibilizado, há um *bug* que acontece raramente quando, após o usuário preencher corretamente suas credenciais na caixa de autenticação básica fornecida pelo navegador, o servidor envia o erro de "Request Timeout". Nesse sentido, depois de diversas mudanças no código-fonte (como a inserção do cabeçalho "Keep-Alive" que informa o tempo em segundo até que um *timeout* de conexão ocorra e o aumento desse tempo de 2 para 5 segundos), bem como de várias tentativas de teste, tal erro parece ter desaparecido. Entretando, dada a dificuldade de se depurar e compreender exatamente o comportamento de leitura do soquete apresentado pelos *browsers* empregues para teste, não é possível afirmar com certeza que esse problema foi extinto.

4.2) Possíveis aprimoramentos futuros:

Algumas melhorias que poderiam ser aplicadas no código final do programa são as seguintes:

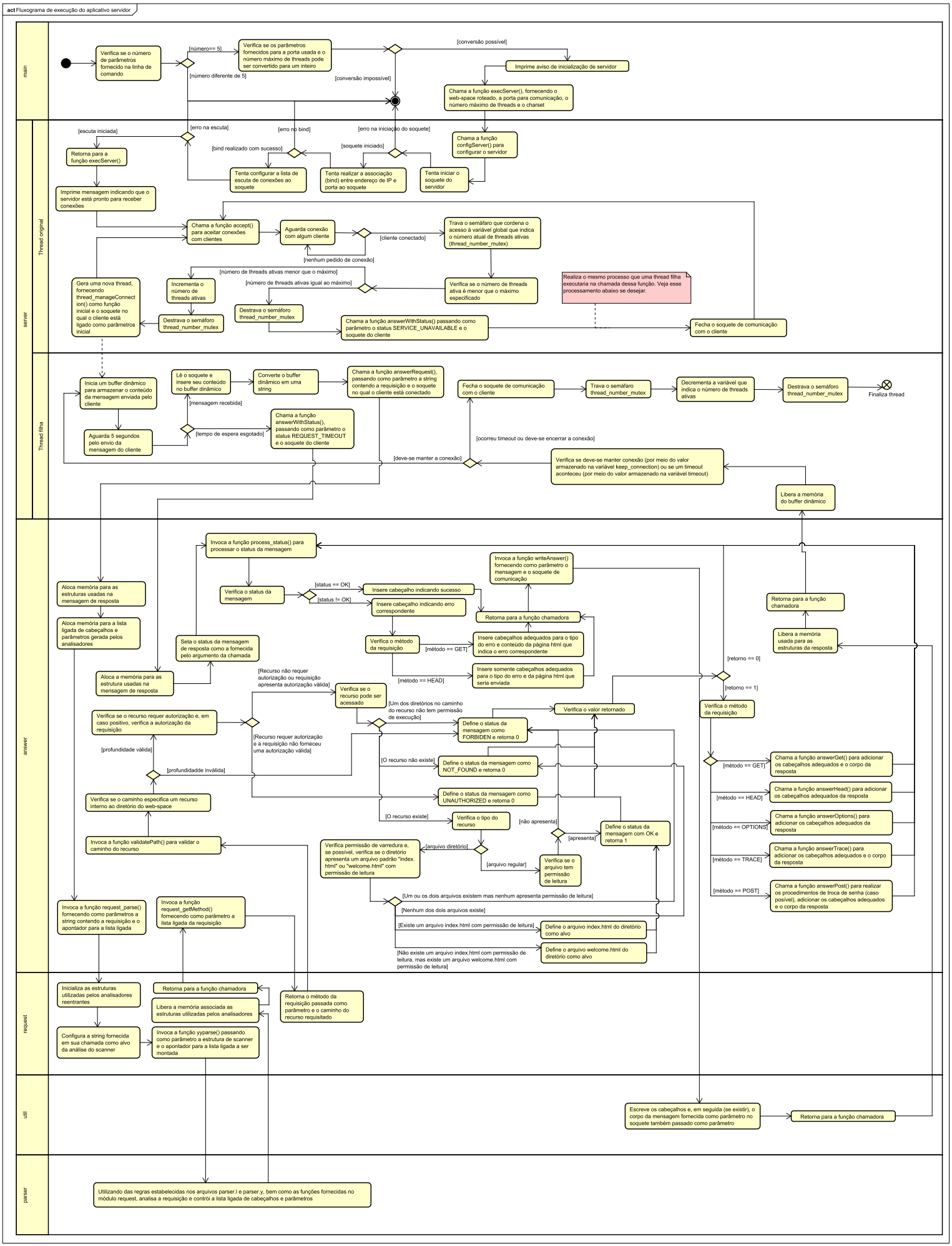
- Refatorar a estrutura desenvolvida até então, de modo a aplicar melhor os conceitos de boas práticas para programação estruturada e aumentar a extensibilidade (facilidade de implementação de novas funcionalidades) do programa final;
- Tanto aprimorar a verificação inicial de parâmetros, quanto inserir impressões no terminal de mensagens que informem quanto aos processamentos realizados pelo aplicativo, de modo a tornar o programa mais responsivo ao usuário;
- Reavaliar a eficiência em quesitos de gasto de memória e velocidade de resposta de requisição, aplicando, se possível, técnicas de programação eficazes que melhorem tais pontos.

4.3) Comentários sobre a disciplina:

A disciplina, no geral, foi produtiva e proporcionou de forma efetiva o aprendizado sobre os diferentes tópicos abordados durante o semestre. Tal fato pode ser percebido ao se realizar uma autoavaliação e comparar o nível de conhecimento atual com aquele apresentado durante o início das aulas. Dessa forma, é possível afirmar que ela cumpriu seu objetivo.

Entretanto, um ponto que poderia ser aprimorado para semestres posteriores é a quantidade e variedade do material de estudo disponibilizado aos alunos, bem como o direcionamento adequado de cada tópico estudado dentro do material fornecido. Tal material poderia abordar não só os tópicos trabalhados durante as aulas, mas também conceitos acerca do desenvolvimento de grandes aplicativos na linguagem C (que costuma ser menos usada nos demais cursos da graduação) de qualidade. Essa proposta tem o intuito tanto de facilitar a obtenção de informações acerca dos diversos padrões que devem ser seguidos nas normas de comunicação sem que seja necessário recorrer aos inúmeros RFCs existentes , bem como de melhorar a qualidade e facilidade do desenvolvimento procedural do projeto final.

5) Apêndice:


act Fluxograma de execução do aplicativo servidor
main
Verifica se o número de parâmetros fornecido na linha de comando
[número== 5]
[número diferente de 5]
Verifica se os parâmetros fornecidos para a porta usada e o número máximo de threads pode ser convertido para um inteiro
[conversão possível]
[conversão impossível]
Imprime aviso de inicialização de servidor
Chama a função execServer(), fornecendo o web-space roteado, a porta para comunicação, o número máximo de threads e o charset
server
Thread original
[escuta iniciada]
[erro na escuta]
[erro no bind]
[erro na iniciação do soquete]
Chama a função configServer() para configurar o servidor
Retorna para a função execServer()
[bind realizado com sucesso]
[soquete iniciado]
Tenta configurar a lista de escuta de conexões ao soquete
Tenta realizar a associação (bind) entre endereço de IP e porta ao soquete
Tenta iniciar o soquete do servidor
Imprime mensagem indicando que o servidor está pronto para receber conexões
Chama a função accept() para aceitar conexões com clientes
Aguarda conexão com algum cliente
[cliente conectado]
Trava o semáfaro que cordena o acesso à variável global que indica o número atual de threads ativas (thread_number_mutex)
[nenhum pedido de conexão]
[número de threads ativas menor que o máximo]
[número de threads ativas igual ao máximo]
Verifica se o número de threads ativa é menor que o máximo especificado
Realiza o mesmo processo que uma thread filha executaria na chamada dessa função. Veja esse processamento abaixo se desejar.
Gera uma nova thread, fornecendo thread_manageConnection() como função inicial e o soquete no qual o cliente está ligado como parâmetros inicial
Incrementa o número de threads ativas
Destrava o semáforo thread_number_mutex
Destrava o semáforo thread_number_mutex
Chama a função answerWithStatus() passando como parâmetro o status SERVICE_UNAVAILABLE e o soquete do cliente
Fecha o soquete de comunicação com o cliente
Thread filha
Inicia um buffer dinâmico para armazenar o conteúdo da mensagem enviada pelo cliente
Lê o soquete e insere seu conteúdo no buffer dinâmico
Converte o buffer dinâmico em uma string
Chama a função answerRequest(), passando como parâmetro a string contendo a requisição e o soquete no qual o cliente está conectado
[mensagem recebida]
Aguarda 5 segundos pelo envio da mensagem do cliente
[tempo de espera esgotado]
Chama a função answerWithStatus(), passando como parâmetro o status REQUEST_TIMEOUT e o soquete do cliente
Fecha o soquete de comunicação com o cliente
Trava o semáfaro thread_number_mutex
Decrementa a variável que indica o número de threads ativas
Destrava o semáforo thread_number_mutex
Finaliza thread
[ocorreu timeout ou deve-se encerrar a conexão]
[deve-se manter a conexão]
Verifica se deve-se manter conexão (por meio do valor armazenado na variável keep_connection) ou se um timeout aconteceu (por meio do valor armazenado na variável timeout)
Libera a memória do buffer dinâmico
answer
Invoca a função process_status() para processar o status da mensagem
Invoca a função writeAnswer() fornecendo como parâmetro o mensagem e o soquete de comunicação
Aloca memória para as estruturas usadas na mensagem de resposta
Verifica o status da mensagem
[status == OK]
[status != OK]
Insere cabeçalho indicando sucesso
Insere cabeçalho indicando erro correspondente
Retorna para a função chamadora
Retorna para a função chamadora
Aloca memória para a lista ligada de cabeçalhos e parâmetros gerada pelos analisadores
Seta o status da mensagem de resposta como a fornecida pelo argumento da chamada
Verifica o método da requisição
[método == GET]
Insere cabeçalhos adequados para o tipo do erro e conteúdo da página html que indica o erro correspondente
[método == HEAD]
Insere somente cabeçalhos adequados para o tipo do erro e da página html que seria enviada
Libera a memória usada para as estruturas da resposta
[retorno == 0]
[retorno == 1]
Aloca a memória para as estrutura usadas na mensagem de resposta
[Recurso não requer autorização ou requisição apresenta autorização válida]
Verifica se o recurso pode ser acessado
[Um dos diretórios no caminho do recurso não tem permissão de execução]
Verifica o valor retornado
Verifica o método da requisição
Verifica se o recurso requer autorização e, em caso positivo, verifica a autorização da requisição
[Recurso requer autorização e a requisição não forneceu uma autorização válida]
Define o status da mensagem como FORBIDEN e retorna 0
[profundidade válida]
[profundidadde inválida]
[O recurso não existe]
Define o status da mensagem como NOT_FOUND e retorna 0
[método == GET]
Chama a função answerGet() para adicionar os cabeçalhos adequados e o corpo da resposta
[método == HEAD]
Chama a função answerHead() para adicionar os cabeçalhos adequados da resposta
[método == OPTIONS]
Chama a função answerOptions() para adicionar os cabeçalhos adequados da resposta
[método == TRACE]
Chama a função answerTrace() para adicionar os cabeçalhos adequados e o corpo da resposta
[método == POST]
Chama a função answerPost() para realizar os procedimentos de troca de senha (caso posível), adicionar os cabeçalhos adequados e o corpo da resposta
Define o status da mensagem como UNAUTHORIZED e retorna 0
Verifica se o caminho especifica um recurso interno ao diretório do web-space
[O recurso existe]
Verifica o tipo do recurso
[não apresenta]
[apresenta]
Define o status da mensagem com OK e retorna 1
Invoca a função validatePath() para validar o caminho do recurso
Verifica permissão de varredura e, se possível, verifica se o diretório apresenta um arquivo padrão "index.html" ou "welcome.html" com permissão de leitura
[arquivo diretório]
[arquivo regular]
Verifica se o arquivo tem permissão de leitura
Invoca a função request_parse() fornecendo como parâmetros a string contendo a requisição e o apontador para a lista ligada
Invoca a função request_getMethod() fornecendo como parâmetro a lista ligada da requisição
[Um ou os dois arquivos existem mas nenhum apresenta permissão de leitura]
[Nenhum dos dois arquivos existe]
[Existe um arquivo index.html com permissão de leitura]
Define o arquivo index.html do diretório como alvo
[Não existe um arquivo index.html com permissão de leitura, mas existe um arquivo welcome.html com permissão de leitura]
Define o arquivo welcome.html do diretório como alvo
request
Inicializa as estruturas utilizadas pelos analisadores reentrantes
Retorna para a função chamadora
Libera a memória associada as estruturas utilizadas pelos analisadores
Retorna o método da requisição passada como parâmetro e o caminho do recurso requisitado
Configura a string fornecida em sua chamada como alvo da análise do scanner
Invoca a função yyparse() passando como parâmetro a estrutura de scanner e o apontador para a lista ligada a ser montada
util
Escreve os cabeçalhos e, em seguida (se existir), o corpo da mensagem fornecida como parâmetro no soquete também passado como parâmetro
Retorna para a função chamadora
parser
Utilizando das regras estabelecidas nos arquivos parser.l e parser.y, bem como as funções fornecidas no módulo request, analisa a requisição e contrói a lista ligada de cabeçalhos e parâmetros