

**Trabalho Final - EA872 - Turma K**  
**Laboratório de Programação de Software Básico**



**Aluno:** Vinícius Esperança Mantovani

**RA:** 247395

**Data:** 06/12/2023

## Descrição Detalhada do Sistema e Fluxogramas:

Inicialmente, vale destacar que este relatório foi feito com base naquilo que se obteve do servidor com parser não reentrante majoritariamente, mas parte do relatório tem adaptações para abranger tanto o caso de parser reentrante como o caso de parser não reentrante. Isso se dá, principalmente nesta seção e, inclusive, em trechos dos fluxogramas. O problema que tornou prudente usar o servidor sem parser reentrante ao invés daquele com parser reentrante está explicado nas últimas três seções deste relatório, principalmente na antepenúltima. No entanto, isso não afeta a explicação desta seção de maneira negativa, apenas necessitando de pequenas adaptações, uma vez que os códigos são exatamente iguais, com exceção das partes adaptadas para o parser reentrante.

Sob análise aprofundada do servidor, possibilitou-se a construção dos diagramas das Figuras 1, 2, 3 e 4, respectivamente:

- Um fluxograma da função “main” do programa, que explica detalhadamente como é iniciado o servidor até a criação de uma thread;
- Um fluxograma da função “initThread” do programa, que explica detalhadamente como é iniciada uma thread de atendimento de requisições até o ponto em que ela chama uma das funções correspondentes aos métodos possíveis de requisição;
- Um fluxograma da função “getRes” do programa, que explica como se dá o processo de resposta a uma requisição do tipo GET;
- Um fluxograma da função “postRes” do programa, que explica como se dá o processo de resposta a uma requisição do tipo POST.

Deve-se perceber que não foram feitos fluxogramas para as demais funções de atendimento de requisições (headRes, traceRes ...). Isso ocorre, por causa do fato de essas funções terem um fluxo muito parecido com o de “getRes” como no caso de “headRes” ou terem o fluxo muito simples como no caso de “traceRes” e “optionsRes”. Ademais, existem as funções “printCode” e “printsAnswer”, cujas funções são, respectivamente, imprimir a primeira linha do cabeçalho, escrevendo o código HTTP passado ao cliente e, imprimir todo o resto do cabeçalho de uma requisição. Essas funções, como “traceRes” e “optionsRes”, têm um fluxo bastante simples, logo também não foram feitos fluxogramas a respeito delas.

Para entender o fluxo do programa e compreender como o servidor funciona de uma maneira geral, basta analisar os diagramas a seguir e entendê-los a fundo. Abaixo dos fluxogramas, está uma explicação a respeito do funcionamento do parser e da estrutura usada para obtenção dos dados vindos do parser.

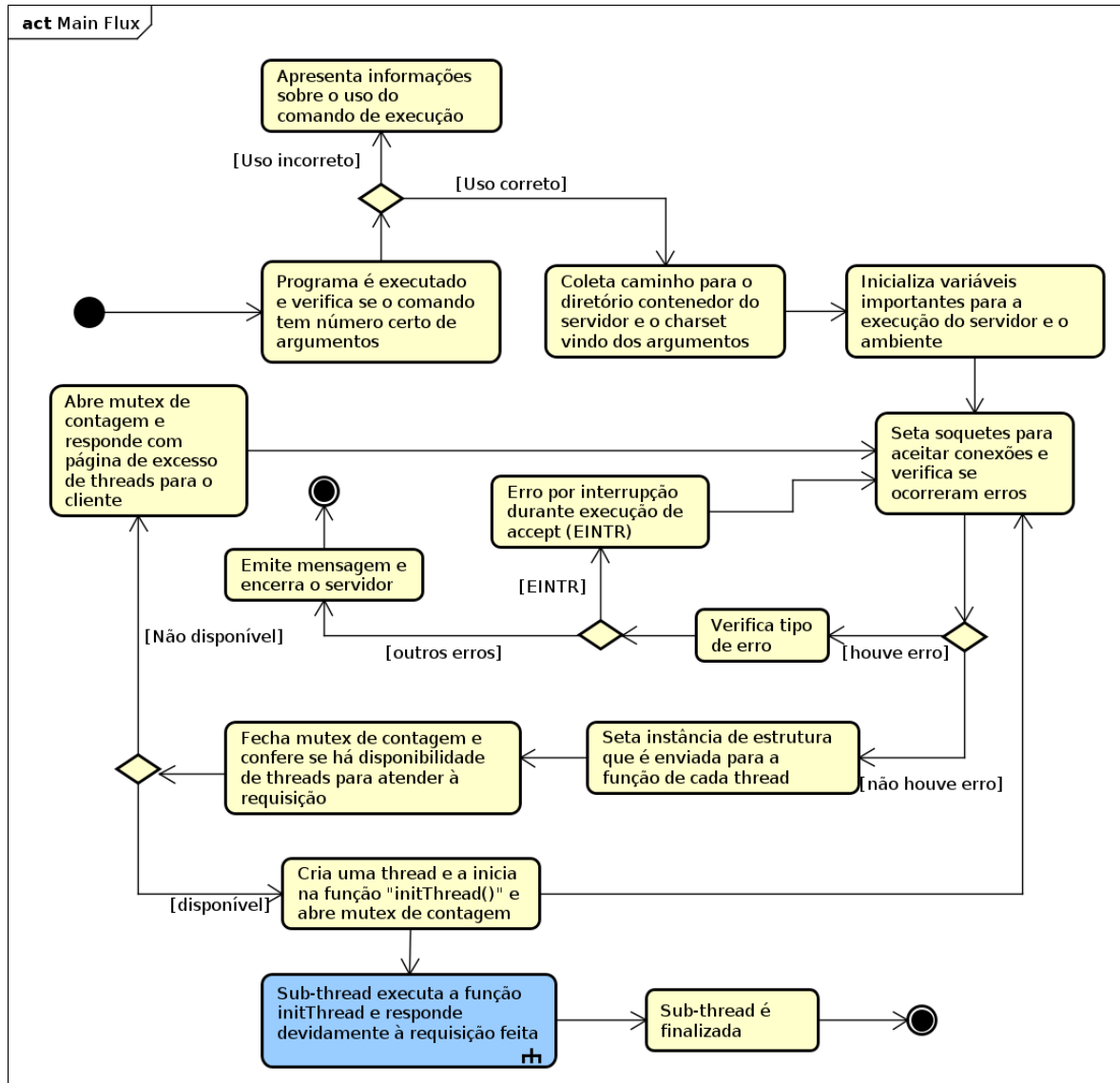


Figura 1: Fluxograma da função “main” do programa

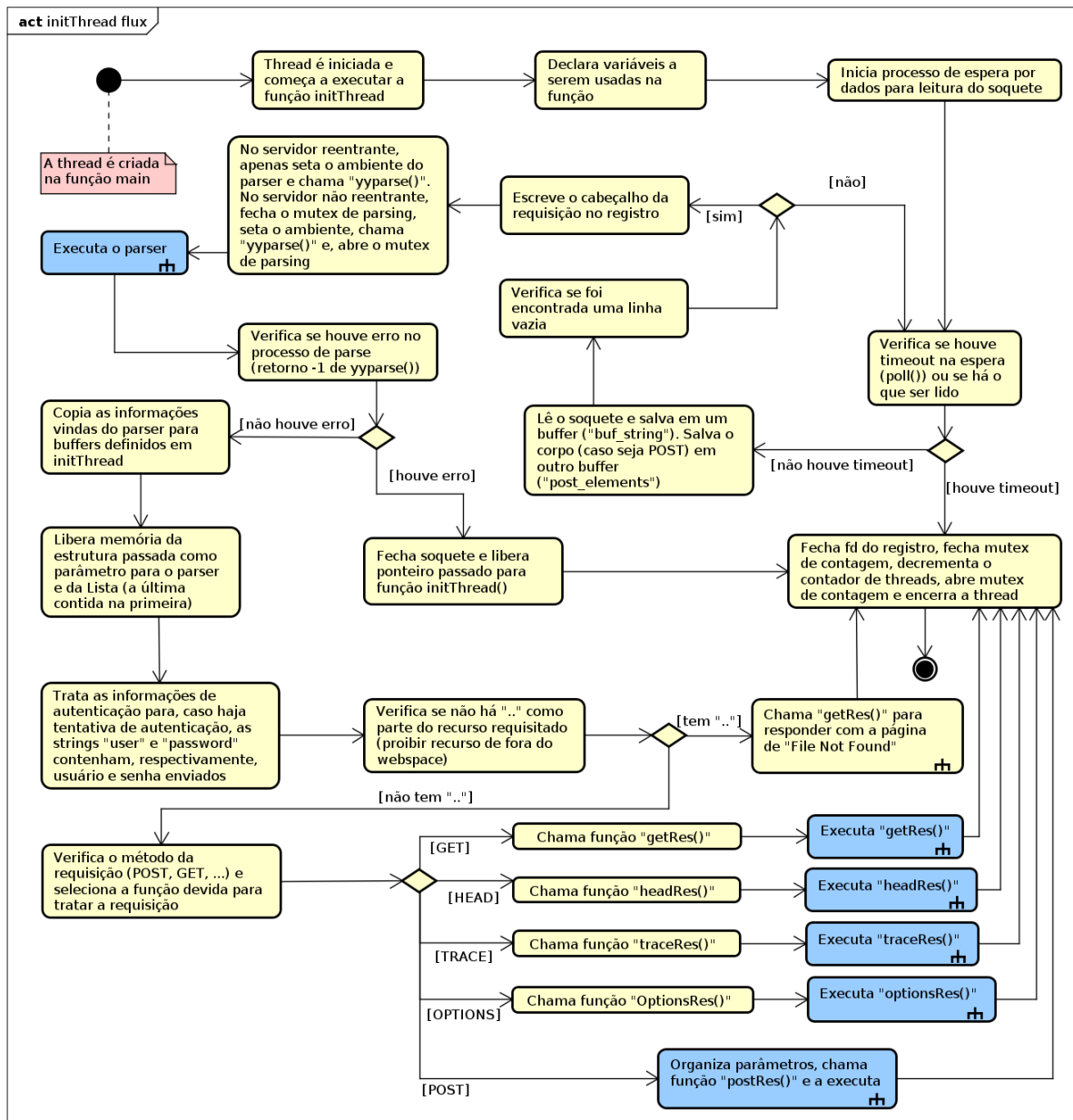


Figura 2: Fluxograma da função "initThread" do programa

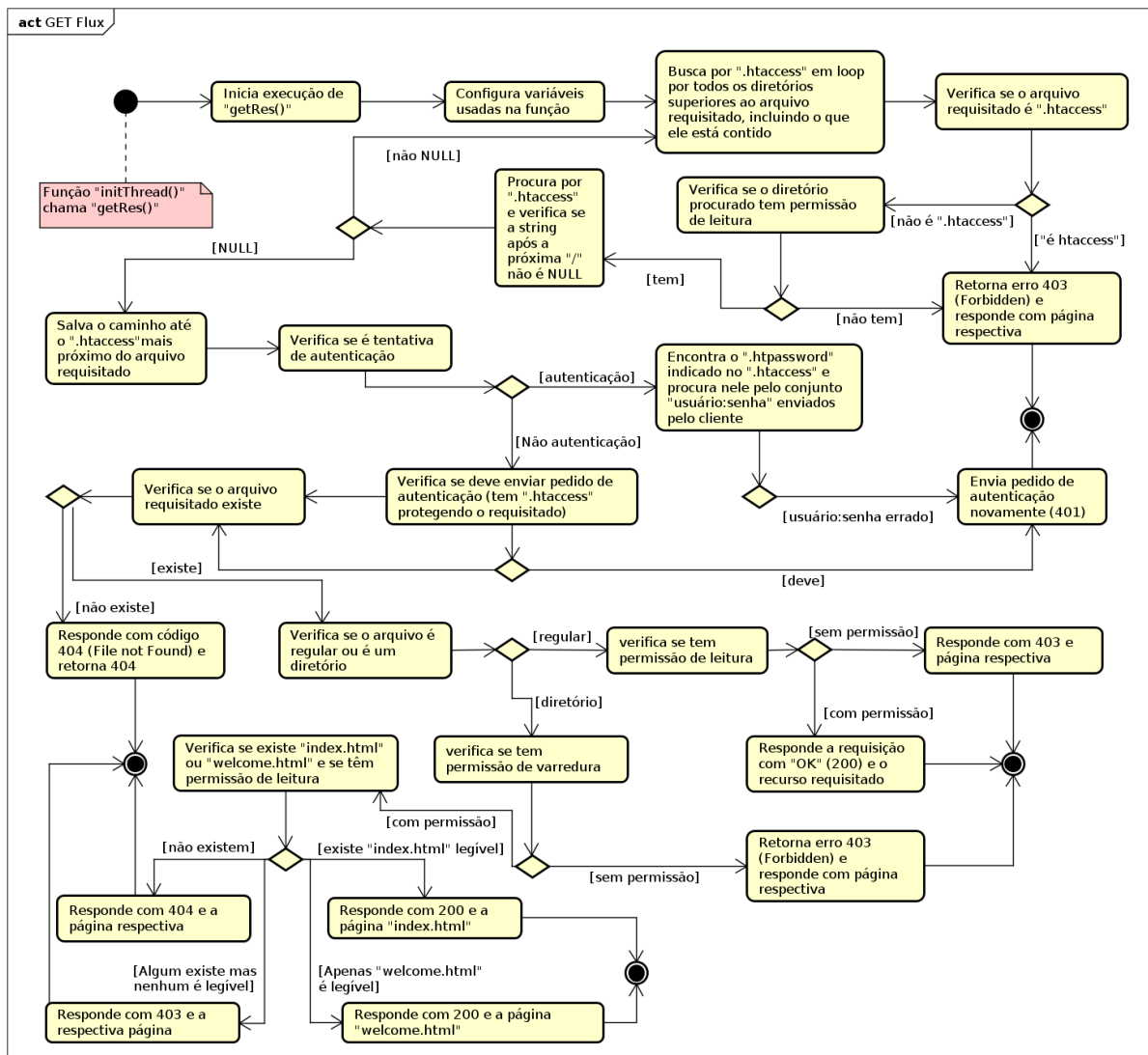


Figura 3: Fluxograma da função "getRes" (GET) do programa

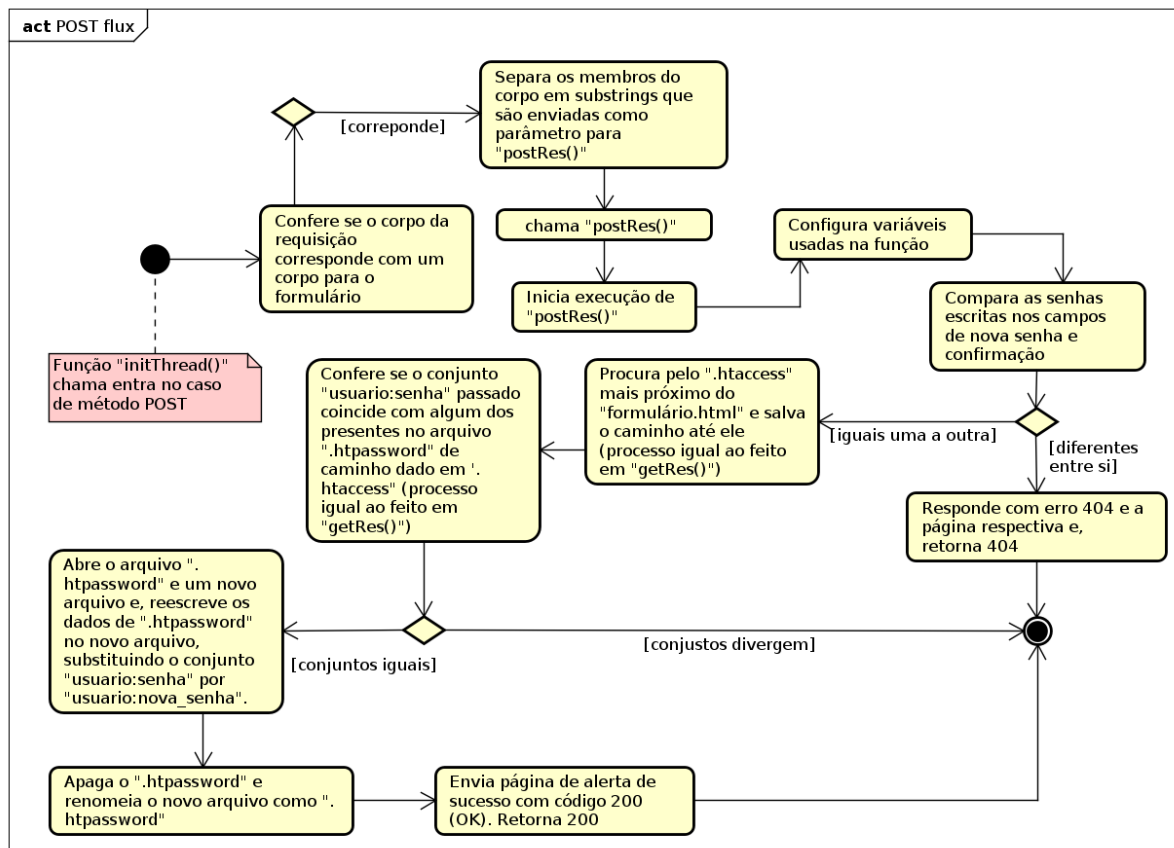


Figura 4: Fluxograma da função “postRes” (POST) do programa

Tratando do parser, cabe iniciar por uma explicação das estruturas usadas para seu correto funcionamento. Começando pela lista ligada: é uma estrutura composta por nós ligados entre si de maneira que cada nó está ligado a apenas um outro e não há ciclos. Desse modo, de maneira mais próxima do código, a lista ligada é uma estrutura chamada de “Lista”, contida no arquivo “environment.h”, que tem três atributos:

- proximo: atributo de tipo ponteiro de Lista que aponta para o próximo nó da lista;
- params: atributo de tipo ponteiro de Lista que aponta para o primeiro nó de uma outra lista;
- comando\_param: atributo de tipo String;

Cada um desses atributos tem uma aplicação importante no armazenamento de informações obtidas durante o processo de parsing. Isso porque, é instanciado inicialmente um elemento do tipo Lista, que é responsável por armazenar os comandos da requisição HTTP e, esta lista armazena, em cada um de seus, uma outra lista que armazena os parâmetros daquele argumento. Sendo assim, “proximo” é o que torna uma instância de “Lista” uma lista de fato, pois é “proximo” que liga um nó ao próximo nó, “params” é responsável por apontar para o início de uma outra lista que contém os parâmetros do comando atual e, por fim, “comando\_param” armazena o comando ou o parâmetro em si (os nomes deles).

Tratando, agora, da estrutura que é passada de parâmetro para o parser, deve-se destacar que ela existe para substituir variáveis que eram antes compartilhadas entre os arquivos do programa de forma global. Para tanto, tem os seguintes atributos:

- Comeco: atributo do tipo ponteiro de Lista, usado para apontar ao começo da lista de comandos de uma requisição (a que está sendo tratada pela thread que chama o parser);
- count\_com e count\_par: inteiros que contam o número de comandos e parâmetros da requisição HTTP, importantes para a lógica de alocação de memória no parser;
- reqs e ress: strings que armazenam respectivamente o recurso requisitado, por exemplo “/dir/index.html” e, o método da requisição, exemplo “GET”.

Com essas duas estruturas, o programa é capaz de obter as informações necessárias do parser e armazená-las para usá-las da maneira que forem necessárias durante a execução do servidor.

## **Conteúdo do webspace usado para testes:**

Nesta seção, tratar-se-á do webspace e de seu conteúdo. Para tanto, é de enorme ajuda analisar a Figura 1, que contém um diagrama de componentes que descreve exatamente a composição do webspace usado para os testes.

Cabe ressaltar que o diagrama de componentes apresentado (Figura 1) apresenta, também, as permissões de cada um dos componentes, sejam eles arquivos regulares ou diretórios. Assim, serve como um bom material de consulta para casos de dúvidas a respeito dos testes.

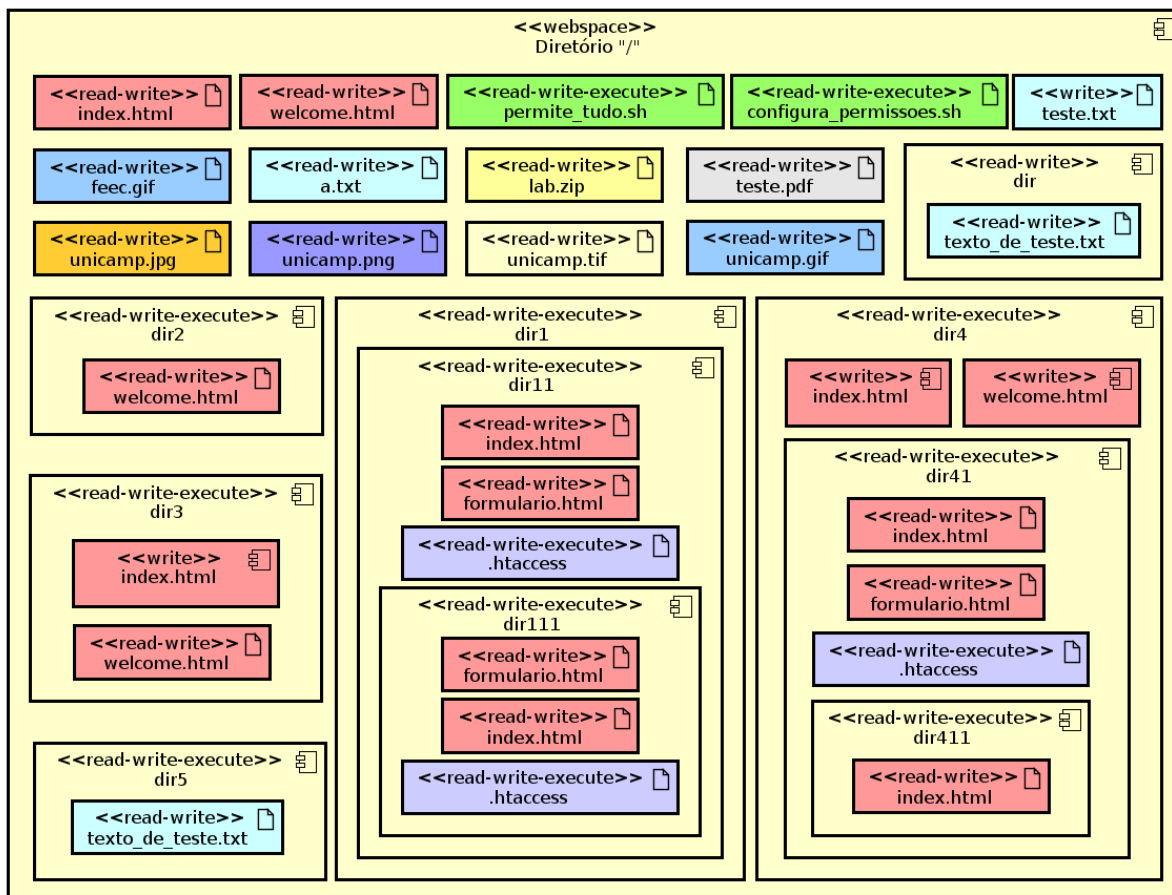


Figura 5: diagrama de componentes do webservice usado para testes

Conforme se pode perceber no diagrama, o webservice tem vários arquivos diretamente em sua raiz. Dentre tais arquivos, tem-se onze arquivos regulares, que são, em sua maioria, utilizados para testes de tipos de arquivos aceitos pelo servidor. No entanto, dois desses arquivos são scripts “bash”, sendo eles:

- permite\_tudo.sh: script usado para precaução contra possíveis perdas de arquivos durante a compressão. Dá todas as permissões para todos os arquivos contidos no webservice recursivamente.
- configura\_permissoes.sh: configura de maneira adequada as permissões de cada um dos arquivos do webservice de maneira a deixá-lo exatamente como deve estar para os testes e como estão representados no diagrama (Figura 1).

Ademais, tem-se vários diretórios usados para diferentes fins que serão melhor esclarecidos na seção de testes, porém, cabe aqui notar, ainda, que existem três arquivos “.htaccess”. Esses arquivos são, também, utilizados para testes e têm como conteúdo o seguinte:

```
webservice_para_testes > dir1 > dir11 > dir111 > .htaccess
1  ../.htpassword1
```



Figura 6: Conteúdo “.htaccess” de “dir1/dir11/dir111”

```
webpace_para_testes > dir1 > dir11 > ⚙ .htaccess
1  ../htpassword2
```

Figura 7: Conteúdo “.htaccess” de “dir1/dir11”

```
webpace_para_testes > dir4 > dir41 > ⚙ .htaccess
1  ../htpassword3
```

Figura 8: Conteúdo “.htaccess” de “dir4/dir41”

Continuando a análise, cada um desses conteúdos é o caminho relativo para um arquivo de senhas (“.htpassword”) relativo ao “.htaccess” que contém seu caminho. Logo, deve-se conhecer o conteúdo desses arquivos de senha para conseguir entender os testes feitos a respeito dos formulários de troca de senha e da autenticação em algumas situações. Por isso, a seguir estão os conteúdos de cada um desses arquivos de senha, ordenados de maneira a seguirem a ordem em que seus respectivos arquivos “.htaccess” são apresentados acima:

```
servidor_threads_nao_reentrante > ≡ .htpassword1
1  aaa:EA4415vStYEKU
2  bbb:EAE/I0MY1s0Y2
3  ccc:EAIUKLHKTeb36
4  abcd:EAjBbqvqe1b96
5  abcd1234:EAM.ymB5MBCzU
```

Figura 9: conteúdo do arquivo de senhas “.htpassword1”

```
servidor_threads_nao_reentrante > ≡ .htpassword2
1  chique:EAd0Bi3Qv5yDc
2  interessante:EAdAJE7GGwvM
3  maravilhoso:EAKooYhW207yY
4  num1234:EAEJYBm5T0Tn.
5  nums0000:EAxzydiLTg3xg
```

Figura 10: conteúdo do arquivo de senhas “.htpassword2”

```
servidor_threads_nao_reentrante > ≡ .htpassword3
1  azul:EA0FcxbYN0jhs
2  amarelo:EAKpsT1xFehQo
3  verde:EA/Q9kba6WW2c
4  vermelho:EAYL1lRq0uynw
5  rosa:EAJWBAokh1.u2
```

Figura 11: conteúdo do arquivo de senhas “.htpassword3”

Com isso, tendo em vista a simplicidade de se entender o webservice por meio do diagrama da Figura 1, não são necessárias mais explicações que precedam os testes para que o entendimento destes seja possível. Logo, siga-se aos testes.

## Testes de funcionalidades:

Atenção: o comando “./configura\_permissoes.sh” tem de ser executado para que o webservice funcione corretamente para os testes!

Inicialmente, vale reconhecer os links da página html principal do webservice que serão usados para os testes. Esses links estão apresentados nas Figuras 12, 13 e 14 abaixo:



Figura 12: Links de página principal 3

### Casos especiais:

- [Arquivo sem permissão de leitura](#)
- [Arquivo inexistente](#)
- [Diretório sem permissão de execução](#)
- [Arquivo dentro do diretório acima que está sem permissão de execução](#)
- [Diretório com index.html](#)
- [Diretório com index.html e protegido por .htaccess.](#)
- [Diretório com index.html e protegido por .htaccess distinto do anterior.](#)
- [Diretório com welcome.html](#)
- [Diretório com index.html sem permissão de leitura, mas com welcome.html legível.](#)
- [Diretório com index.html e welcome.html sem permissão de leitura](#)
- [Diretório com index.html e protegido por .htaccess.](#)
- [Diretório com index.html, sem .htaccess, mas protegido pelo .htaccess do nível de cima.](#)
- [Diretório sem index ou welcome](#)

Figura 13: Links de página principal 2

- [FORMULÁRIO DIR1/DIR11](#)
- [FORMULÁRIO DIR1/DIR11/DIR111](#)
- [FORMULÁRIO DIR4/DIR41](#)

Figura 14: Links de página principal 3

As Figuras estão ordenadas de maneira a manter a ordem dos links que se apresentam na página do webspace. Esta é, também, a ordem em que os testes foram feitos e estão apresentados a seguir.

Nesta seção, estão apresentados os testes realizados no servidor (resultados iguais para os casos com parser reentrante e sem parser reentrante). Siga-se, então, para tais testes:

#### Teste de formatos de arquivos:

O servidor foi desenvolvido de maneira a suportar os tipos “pdf”, “html”, “txt”, “gif”, “jpg” e “png”. Para garantia disso, foram feitos testes, todos bem sucedidos e apresentados a seguir:

Testes do tipo “gif”:

Primeiro link “Formato gif” da Figura 12:

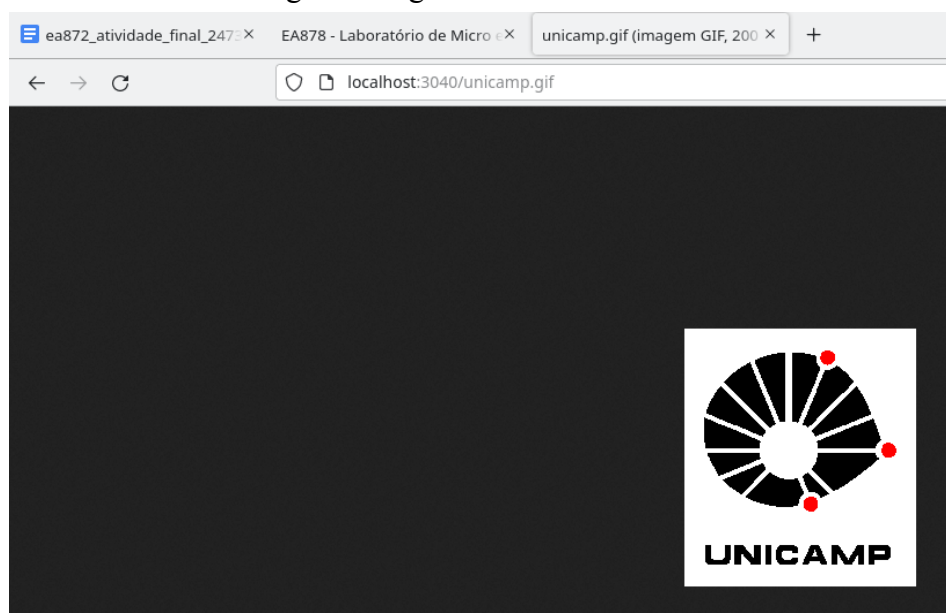


Figura 15: teste “.gif” 1

Segundo link “Formato gif” da Figura 12:



Figura 16: teste “.gif” 2

Testes do tipo “png”:

Link “Formato png” da Figura 12:

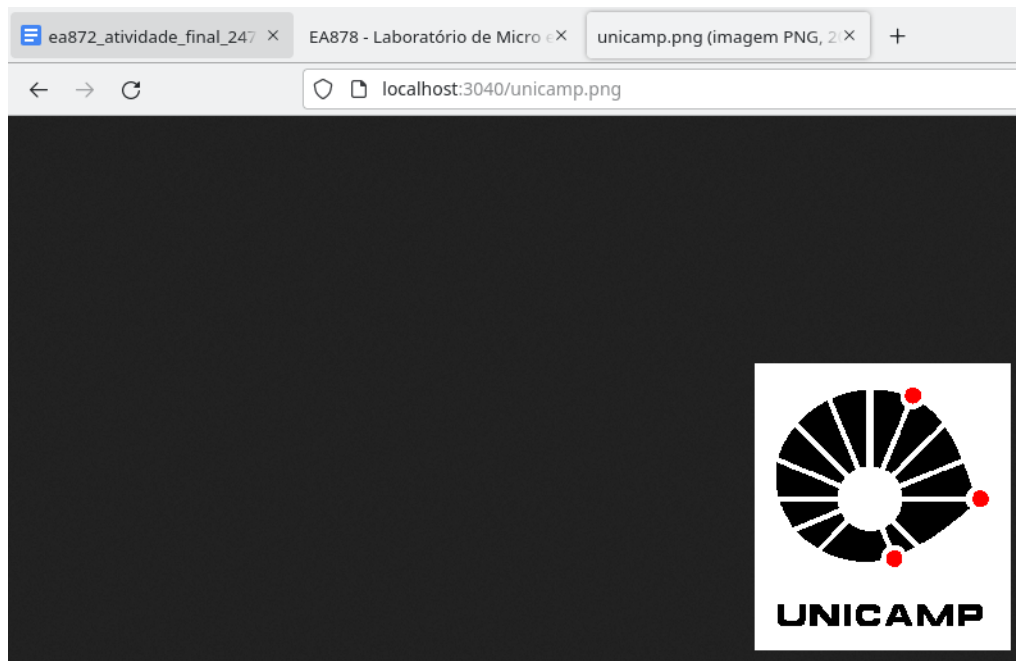


Figura 17: teste “.png”

Testes do tipo “jpg”:

Link “Formato jpg” da Figura 12:

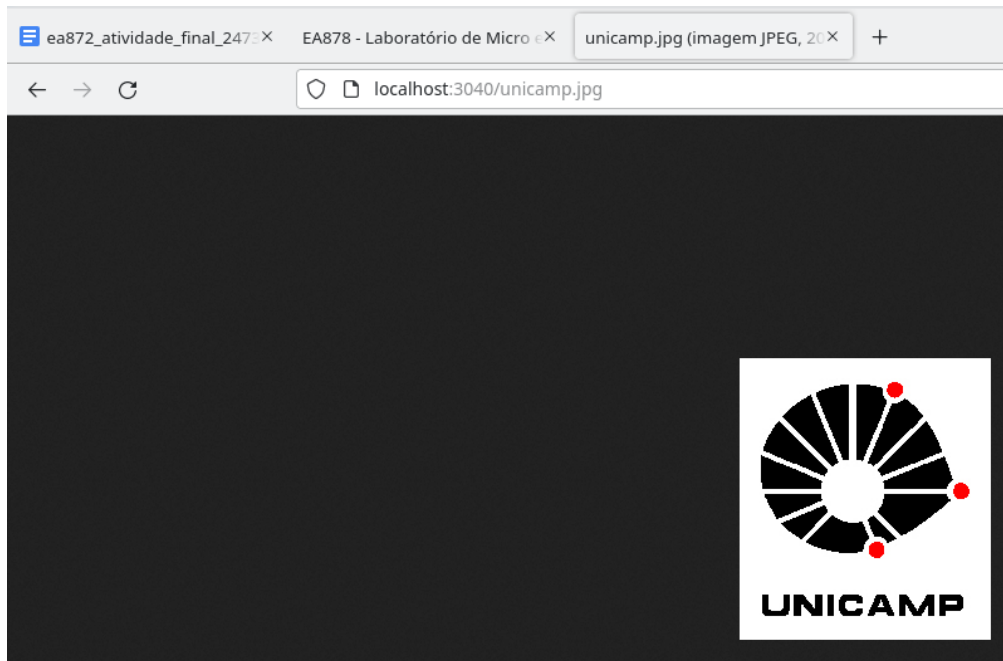


Figura 18: teste “.jpg”

Testes do tipo “pdf”:

Link “Arquivo pdf” da Figura 12:

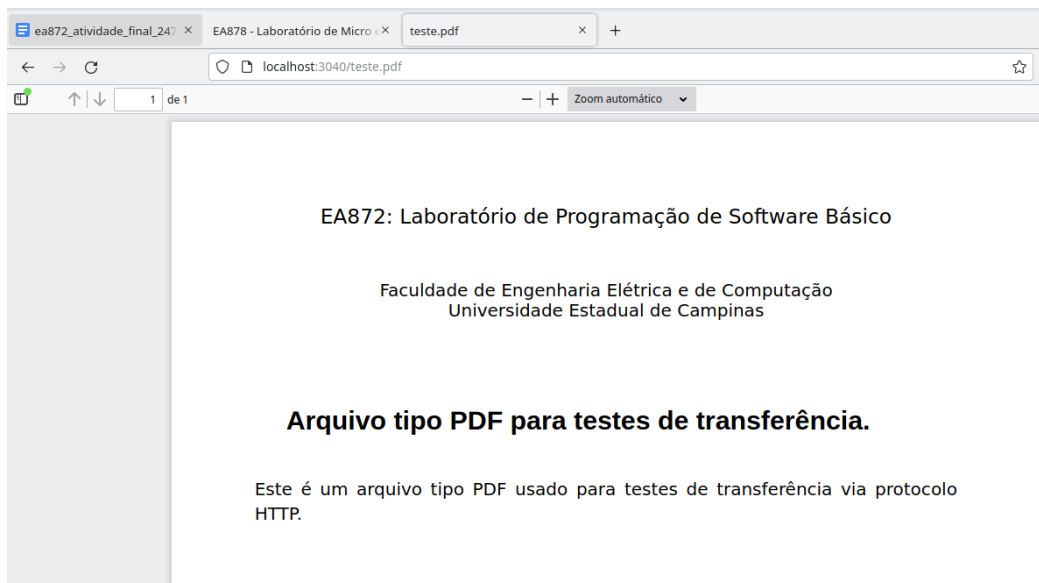


Figura 19: teste “.pdf”

Testes do tipo “txt”:

Link “Arquivo txt” da Figura 12:

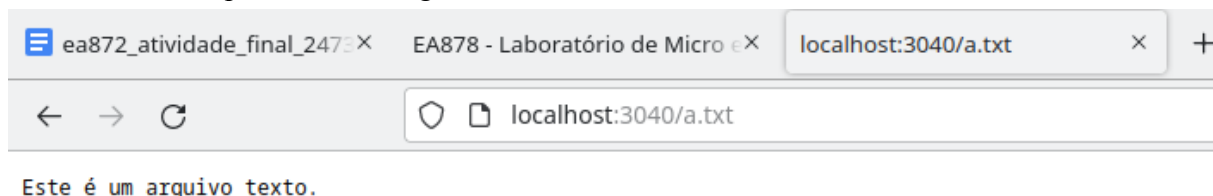


Figura 20: teste “.txt”

Testes do tipo “html”:

Link “Arquivo html” da Figura 12:

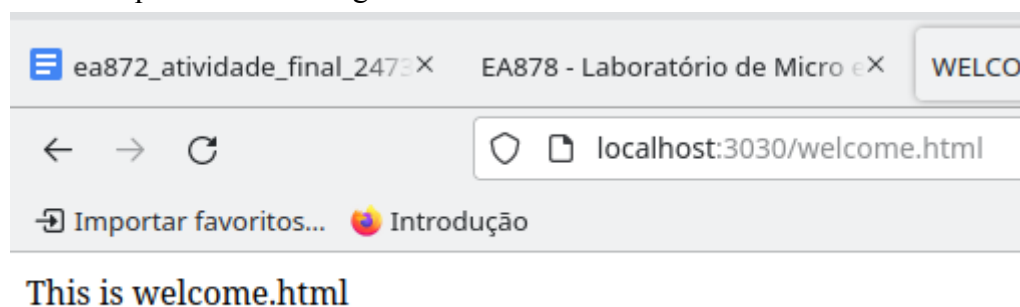


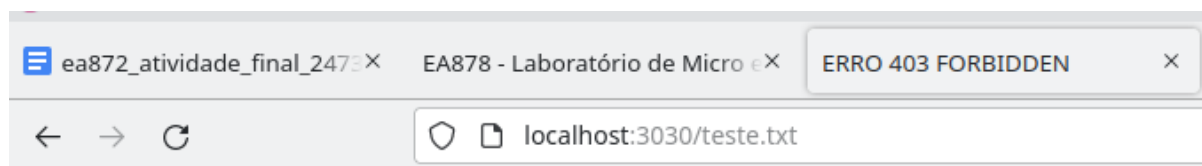
Figura 21: teste “.html”

Como se percebe analisando os resultados amostrados acima, os testes foram bem sucedidos e, portanto, o servidor está lidando corretamente com diferentes tipos de arquivos.

Seguindo, são apresentados os testes de resposta a requisições em diversas situações, links da Figura 13. Estes testes são acompanhados de explicações de como foram feitos e de que estão tratando (embora possa ser um tanto intuitivo por conta dos textos dos links usados). Nos testes abaixo, é importante reconhecer as informações passadas na seção anterior, que trata do web space, de seus arquivos e das permissões de cada um deles. Isso porque, essas informações são de suma importância para a compreensão de cada teste. Logo, caso seja necessário, é importante consultar a seção anterior para esclarecer dúvidas a respeito das permissões de um eventual arquivo usado nos testes que possa gerar dúvidas.

Testes de resposta a requisição por um arquivo sem permissão de leitura:

Este teste foi feito com o arquivo “teste.txt” que se encontra na raiz do web space e não tem permissão de leitura e, ele foi acessado por meio do primeiro link da Figura 13. Resultado:



Um erro foi identificado: O arquivo nao possui permissao para leitura!

Figura 22: resposta a requisição por arquivo sem permissão de leitura

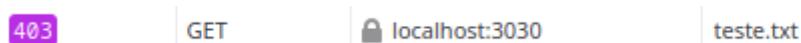
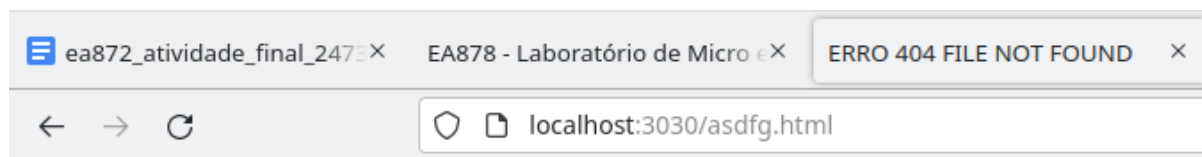


Figura 23: código de erro enviado ao servidor para requisição por arquivo regular sem permissão de leitura

Teste de resposta a requisição por um arquivo inexistente:

O próximo teste apresentado, foi feito com uma tentativa de acessar um arquivo que não existe. Para isso, foi usado o segundo link da Figura 13 e, como se pode perceber, obteve-se sucesso na resposta a tal requisição, tendo o código “404” e a página de “File not Found” retornados:



Um erro foi identificado: O arquivo requisitado não existe!

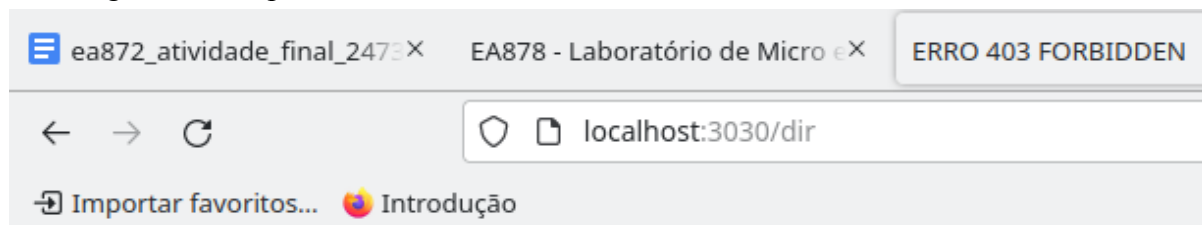
Figura 13: resposta a requisição por arquivo inexistente

404	GET	localhost:3030	asdfg.html
-----	-----	----------------	------------

Figura 24: código de erro enviado ao servidor para requisição por arquivo inexistente

Teste de resposta a requisição por um diretório sem permissão de execução:

No teste abaixo, foi usado o terceiro link da Figura 13 e, é possível verificar que o servidor funciona bem para responder a uma requisição por um diretório sem permissão de varredura (dir, contido na raiz do webspace)(execução), retornando a página de “Forbidden” e o código “403” respectivo:



Um erro foi identificado: O arquivo nao possui permissao para leitura!

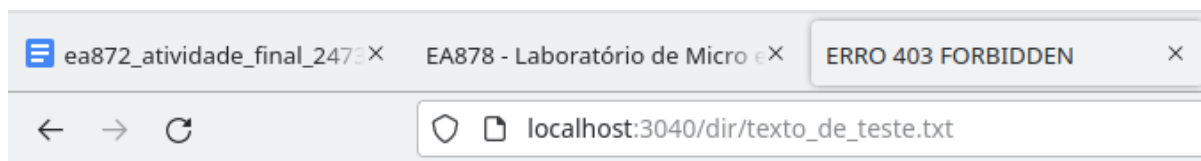
Figura 13: resposta a requisição por diretório sem permissão de execução

403	GET	localhost:3030	dir
-----	-----	----------------	-----

Figura 25: código de erro enviado ao servidor para requisição por diretório sem permissão de leitura

Teste de resposta a requisição por um arquivo cujo diretório acima não tem permissão de execução:

Neste teste, foi utilizado o quarto link da Figura 13 (dir/texto\_de\_teste.txt) e, do mesmo modo como nos casos de requisição por um arquivo sem permissão de leitura e de requisição por um diretório sem permissão de execução, o caso de teste a seguir amostra o tratamento da requisição em questão com o código “403” e a página de “Forbidden”:



Um erro foi identificado: O arquivo nao possui permissao para leitura!

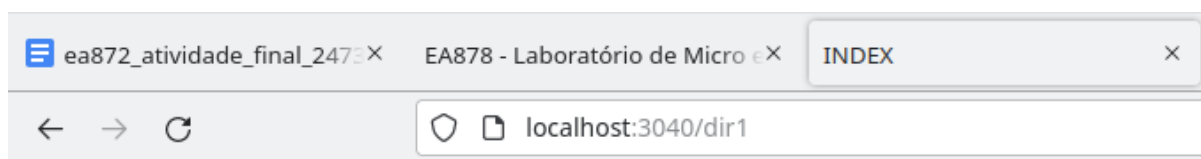
Figura 26: resposta a requisição por arquivo cujo diretório superior não tem permissão de execução

403	GET	localhost:3030	texto_de_teste.txt
-----	-----	----------------	--------------------

Figura 27: código de erro enviado ao servidor para requisição por arquivo cujo diretório superior não tem permissão de leitura

Teste de resposta a requisição por um diretório que contém index.html legível:

No teste que se segue, foi usado o quinto link da Figura 13 e, por consequência, o diretório “dir1” para verificar o funcionamento do servidor em resposta a um diretório que contém “index.html” legível. O resultado, novamente, foi positivo:



This is index.html

Figura 28: resposta a requisição por diretório contenedor de “index.html”

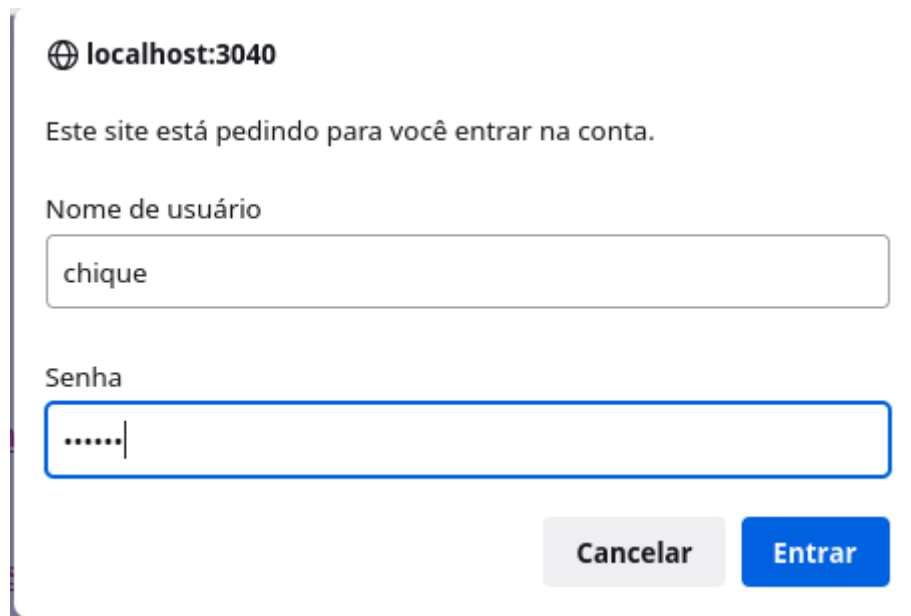
200	GET	localhost:3030	dir1
-----	-----	----------------	------

Figura 29: código de resposta de sucesso em requisição por diretório contenedor de “index.html”

Teste de requisição por um diretório com “index.html” com proteção por “.htaccess”:

Neste teste, foi utilizado o sexto link da Figura 13, que leva ao diretório “dir11” contido em “dir1”, que por sua vez está contido na raiz do webspace. Este diretório contém um arquivo “.htaccess”, que o protege e protege, também aos arquivos internos a ele. Desse modo, testou-se tanto a resposta à requisição pelo arquivo, como a autenticação, conforme segue:





localhost:3040

Este site está pedindo para você entrar na conta.

Nome de usuário

Senha

Cancelar Entrar

Figura 30: resposta imediata do navegador pedindo autenticação como consequência do pedido do servidor (usuário:senha = chique:chique)

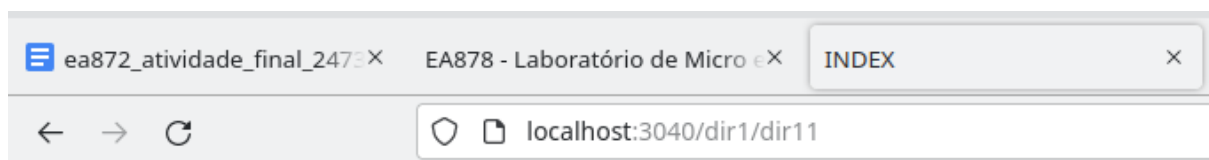
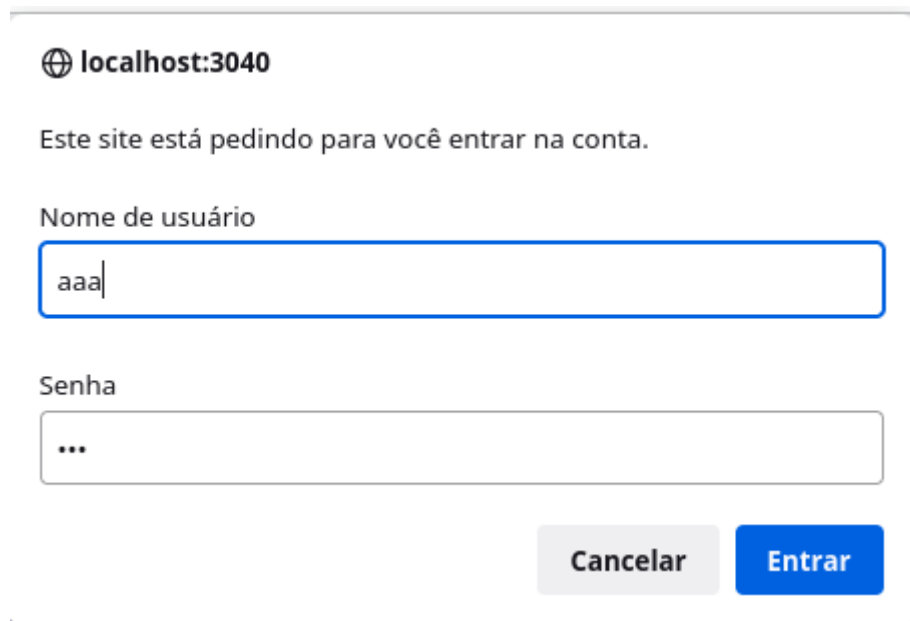


Figura 31: resposta do servidor após autenticação

Vale ressaltar que no teste acima, não foi possível conferir o código enviado ao navegador, pois este não aparece enquanto não é feita autenticação. No entanto, é bastante óbvio que foi “401”, uma vez que somente neste caso o navegador pediria autenticação ao usuário.

Seguindo neste teste, fez-se um teste de uma requisição por um diretório contido em “dir11” mas com outro “.htaccess”. Este diretório é o “dir111” e, o resultado obtido foi o seguinte (sétimo link da Figura 13):



localhost:3040

Este site está pedindo para você entrar na conta.

Nome de usuário

aaa

Senha

...

Cancelar Entrar

Figura 32: resposta imediata do navegador pedindo autenticação como consequência do pedido do servidor (usuário:senha = aaa:aaa)

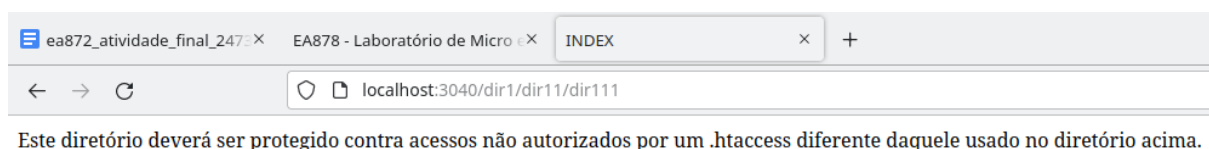
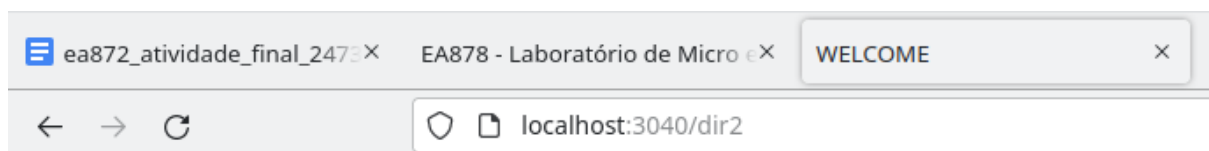


Figura 33: resposta do servidor após autenticação

Vale notar que, no teste acima, como se esperava, o servidor pediu autenticação do usuário novamente quando ocorreu a requisição por um diretório que continha um “.htaccess” diferente do diretório anterior. Isso se dá, porque o servidor considera o “.htaccess” mais próximo do arquivo requisitado e, neste caso, o “.htaccess” mais próximo de “dir11” não é o mesmo do de “dir11”.

Teste de requisição por diretório contenedor apenas de “welcome.html”:

No teste seguinte, foi usado o diretório “dir2”, por meio do oitavo link da Figura 13, para testar se o servidor se comporta corretamente com um diretório contendo apenas “welcome.html” legível:



This is welcome.html

Figura 34: resposta a requisição por diretório contendo “welcome.html” legível e não “index.html”

200	GET	localhost:3133	dir2
-----	-----	----------------	------

Figura 35: código de resposta de sucesso em requisição por diretório contenedor de “welcome.html”

Teste de diretório com “index.html” sem permissão de leitura e “welcome.html” com:

O teste a seguir foi feito usando o nono link Figura 13 que requisita o diretório “dir3”, desse modo, foi possível analisar o comportamento do servidor em resposta a um diretório que contém os arquivos informados acima. O servidor se mostrou bem sucedido, como se nota a seguir:

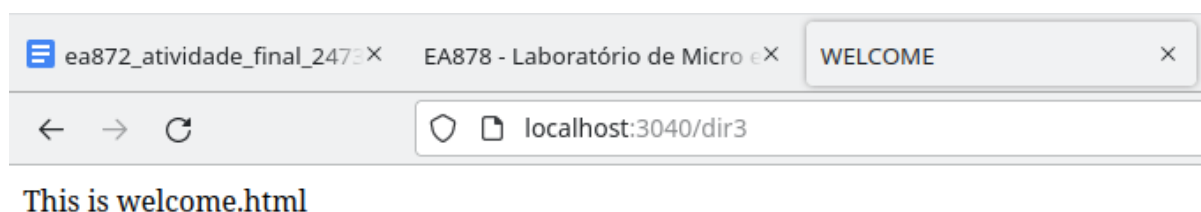


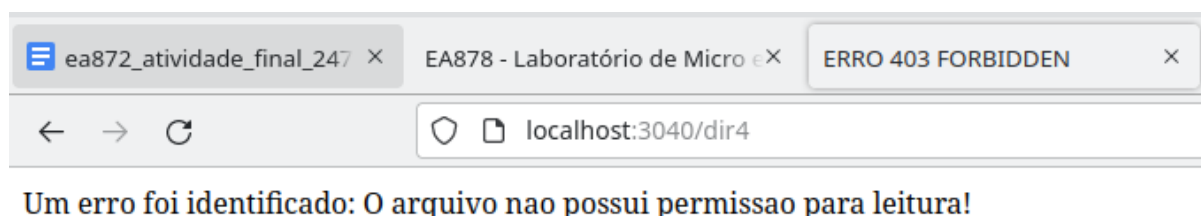
Figura 36: resposta a requisição por diretório contendo “welcome.html” legível e “index.html” sem permissão de leitura

200	GET	localhost:3133	dir3
-----	-----	----------------	------

Figura 37: código de resposta de sucesso em requisição por diretório contenedor de “welcome.html” legível e “index.html” sem permissão de leitura

Teste de requisição por diretório com “index.html” e “welcome.html” ambos sem permissão de leitura:

Neste teste, foi usado o décimo link da Figura 13, que requisita o diretório “dir4”, contido na raiz do webspace. Esse diretório contém ambos os arquivos nas condições citadas acima, permitindo a verificação da resposta do servidor para tal caso. Conforme o esperado, temos erro “403” (Forbidden) e a página referente a ele é enviada:



Um erro foi identificado: O arquivo nao possui permissao para leitura!

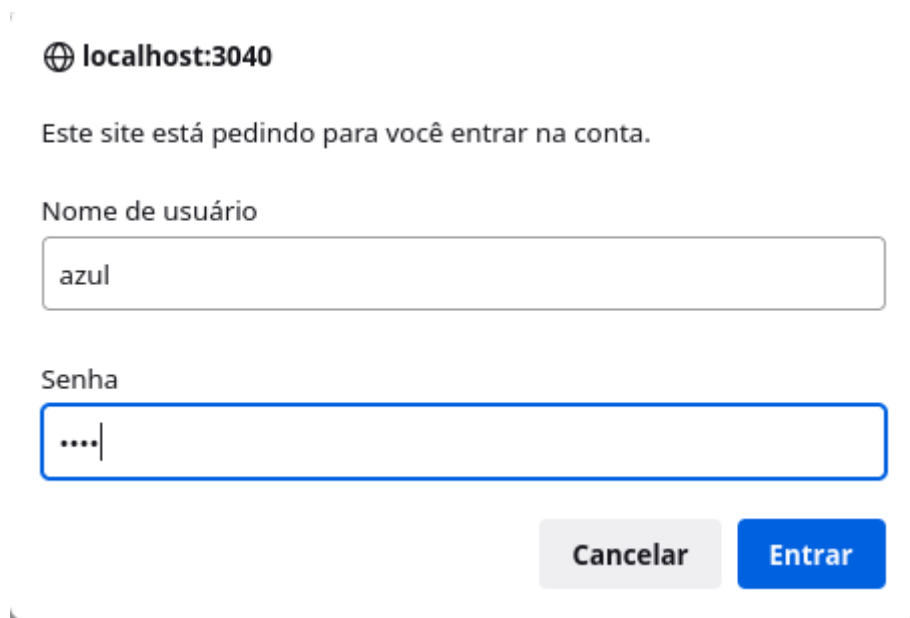
Figura 38: resposta a requisição por diretório contendo “welcome.html” e “index.html” sem permissão de leitura

403	GET	localhost:3133	dir4
-----	-----	----------------	------

Figura 39: código de erro enviado ao servidor para requisição por diretório contendo “welcome.html” e “index.html” sem permissão de leitura

Teste de requisição por diretório com “index.html” protegido por “.htaccess”:

O teste que se segue foi realizado com o antepenúltimo link da Figura 13 (“dir41”, caminho “dir4/dir41”) e visa a conferir o funcionamento do servidor no caso em que temos um arquivo protegido por “.htaccess”. Assim, foi possível verificar que o servidor está funcionando normalmente tanto para pedir a autenticação como para aceitá-la:



localhost:3040

Este site está pedindo para você entrar na conta.

Nome de usuário

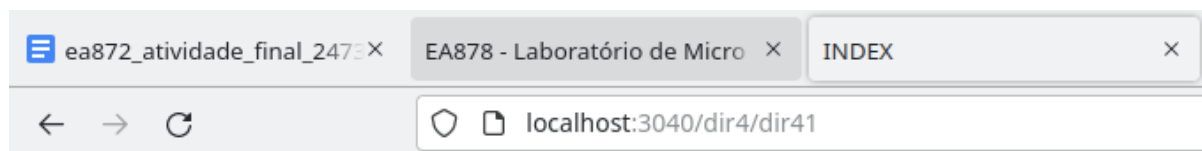
azul

Senha

....

Cancelar Entrar

Figura 40: resposta imediata do navegador pedindo autenticação como consequência do pedido do servidor (usuário:senha = azul:azul)



Este diretório deverá ser protegido por .htaccess contra acessos não autorizados.

Figura 41: resposta após autenticação (diretório contendo “index.html” protegido por “.htaccess”)

Teste de diretório com “index.html” sem “.htaccess” mas protegido pelo “.htaccess” existente no diretório acima:

Para este teste, foi utilizado o penúltimo link da Figura 13, que leva ao diretório “dir411” de caminho “dir4/dir41/dir411”. Isso foi feito como forma de verificar o funcionamento do servidor no caso do requisitado ter em seu caminho algum “.htaccess”, mesmo que não dentro de si próprio, no caso de ser um diretório, ou dentro de seu diretório pai no caso de ser um arquivo regular. Assim, obteve-se o seguinte resultado:

localhost:3133

Este site está pedindo para você entrar na conta.

Nome de usuário

azul

Senha

....

Cancelar Entrar

Figura 42: resposta imediata do navegador pedindo autenticação como consequência do pedido do servidor (usuário:senha = azul:azul)

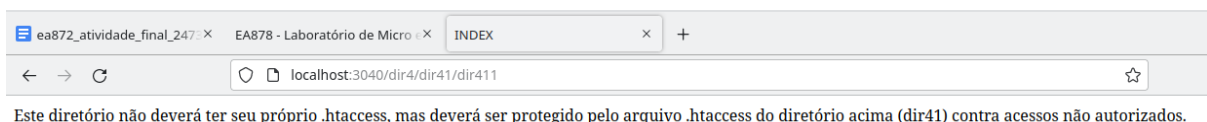


Figura 43: resposta direta do servidor após autenticação

Deve-se notar que a senha usada foi a mesma que aquela usada no teste anterior. Isso é possível, pois como não há “.htaccess” diretamente no diretório requisitado, o servidor busca pelo “.htaccess” mais próximo, que é o do diretório “dir41”, mesmo que do teste anterior. Logo, o arquivo de senhas é também o mesmo.

Teste de diretório sem “index.html” e “welcome.html”:

Para testar isso, foi usado o último link da Figura 13, que requisita o “/dir5” do servidor, diretório nas condições citadas. Assim, é possível conferir que o servidor retorna “File Not Found” (404), conforme deveria. Resultado:

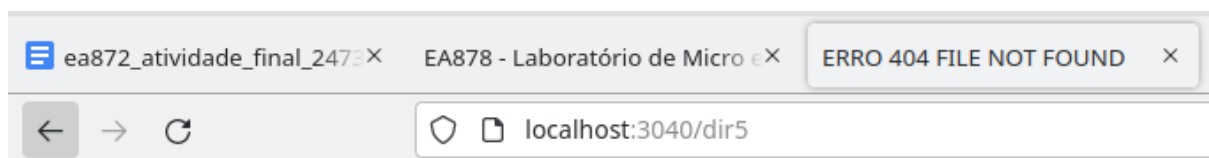


Figura 44: resposta a requisição por diretório sem “index.html” e “welcome.html”

404	GET	localhost:3133	dir5
-----	-----	----------------	------

Figura 45: código de erro enviado ao servidor para requisição por diretório contendo “welcome.html” e “index.html” sem permissão de leitura

Seguindo, são apresentados alguns testes complementares aos testes feitos com links da página principal do webspace:

Teste de requisição por “.htaccess”:

Neste teste, embora a mensagem não seja muito apropriada para o caso, é impressa a página referente ao erro “Forbidden” (403), conforme é esperado:

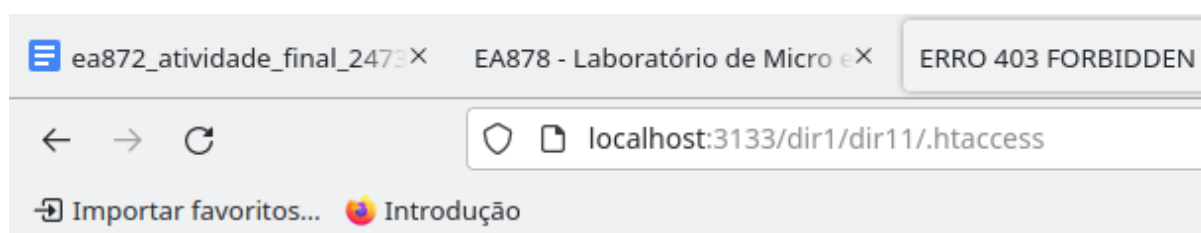


Figura 46: resposta a requisição por “.htaccess”

Vale destacar que o título que aparece na aba do navegador é bastante apropriado, embora a mensagem não seja. Além disso, note que a mensagem não foi projetada para este fim no início da implementação do servidor e foi preservada como estava até o momento, por isso não é apropriada para o caso em questão.

Teste de credenciais incorretas na tentativa de autenticação para requisição por arquivo protegido por “.htaccess”:

Neste teste, não há sentido em apresentar imagens, uma vez que não serviriam de nada. Porém, cabe explicar que, no caso testado, o servidor apenas reenvia o pedido de autenticação (401) e aguarda pelas credenciais novamente.

Por fim, seguindo, tem-se os testes relacionados aos formulários de troca de senha:

Teste de alteração de senha e tentativa de autenticação com a senha antiga após alteração, usando formulário de “dir1/dir11/” e de “dir1/dir11/dir11”:

Como se pode perceber na Figura 10, que apresenta as credenciais criptografadas do arquivo de senha “.htpassword2”, tem-se inicialmente que como um dos conjuntos “usuário:senha” o conjunto “chique:EAdOBi3Qv5yDc” (chique:chique). Além disso, do arquivo “.htpassword1” (Figura 9), tem-se como um dos conjuntos “ccc:EAIUKLHKTEb36” (ccc:ccc). Então, siga-se para a troca de senha:

The screenshot shows a web browser window with two tabs: 'Laboratório de Micro' and 'Atualização de senha do usuário'. The address bar displays 'localhost:3040/dir1/dir11/formulario.html'. The page title is 'EA872 - Laboratório de Programação de Software Básico'. Below the title is the heading 'Formulário para atualizar senha do usuário'. The form contains four input fields: 'Digite seu nome de usuário cadastrado:' with the value 'chique', 'Digite a senha atual:' with masked characters '.....', 'Digite a nova senha (mínimo 8 caracteres):' with masked characters '.....', and 'Confirme a nova senha:' with masked characters '.....'. A button labeled 'Enviar' is positioned below the form fields.

Figura 47: Formulário de troca de senha “dir11” preenchido com as credenciais atuais e nova senha (chique:chique, senha nova: teste)

The screenshot shows a web browser window with two tabs: 'Laboratório de Micro' and 'Atualização de senha do usuário'. The address bar displays 'localhost:3040/dir1/dir11/dir111/formulario.html'. The page title is 'EA872 - Laboratório de Programação de Software Básico'. Below the title is the heading 'Formulário para atualizar senha do usuário'. The form contains four input fields: 'Digite seu nome de usuário cadastrado:' with the value 'ccc', 'Digite a senha atual:' with masked characters '...', 'Digite a nova senha (mínimo 8 caracteres):' with masked characters '.....', and 'Confirme a nova senha:' with masked characters '.....'. A button labeled 'Enviar' is positioned below the form fields.

Figura 48: Formulário de troca de senha “dir111” preenchido com as credenciais atuais e nova senha (ccc:ccc, senha nova: teste)

Em sequência, após o pressionamento do botão “envia” (execução de POST), tem-se o seguinte:

```
.htpassword2 x
servidor_threads_nao_reentrante > .htpassword2
1  cheque:EABjHA72/b9H6
2  interessante:EAaDAJE7GGwvM
3  maravilhoso:EAKooYhW207yY
4  num1234:EAEJYBm5T0Tn.
5  nums0000:EAxzydiLTg3xg
6  |
```

Figura 49: arquivo de senhas “.htpassword2” após alteração de senha do usuário “cheque”

```
functions.c 9+  .htpassword1 x
servidor_threads_nao_reentrante > .htpassword1
1  aaa:EA4415vStYEKU
2  bbb:EAE/I0MYls0Y2
3  ccc:EABjHA72/b9H6
4  abcd:EAbBqvqe1b96
5  abcd1234:EAM.ymB5MBCzU
6  |
```

Figura 50: arquivo de senhas “.htpassword1” após alteração de senha do usuário “ccc”

Assim, pode-se perceber que a senha foi realmente alterada no arquivo de senhas e que as credenciais dos demais usuários permaneceram intactas.

Além disso, é interessante saber que, após uma requisição POST de formulário bem sucedida, o servidor envia a seguinte página para o cliente:

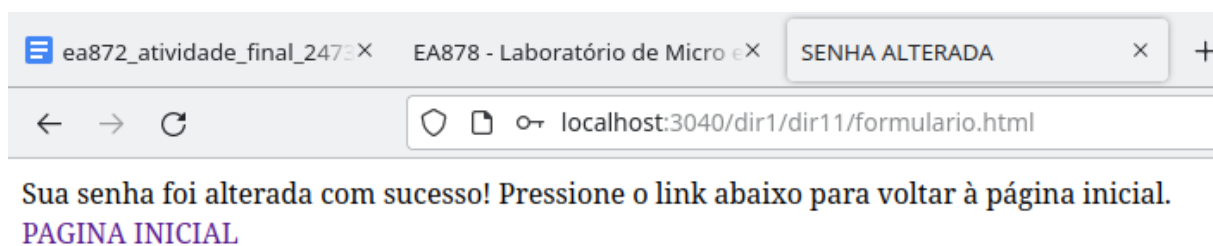


Figura 51: resposta a uma requisição POST de formulário bem sucedida

O botão “PAGINA INICIAL” leva o usuário até a página principal do webspace, que contém os links das Figuras 12, 13 e 14.

Finalmente, o servidor responde da seguinte forma o caso em que o usuário tenta se autenticar com a senha antiga:

**User or Password doesn't match**

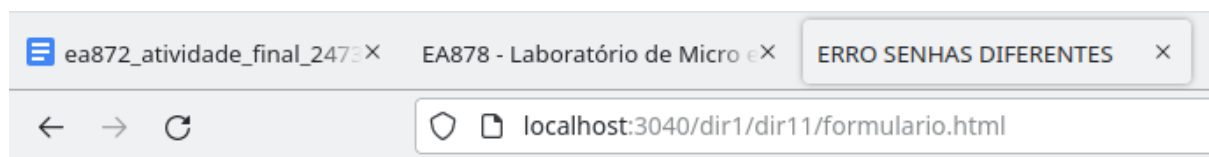


Figura 52: resposta do servidor a uma tentativa de requisição com as credenciais antigas após alteração (resposta no console)

Vale notar que o servidor não responde com uma página para este caso, ele apenas requisita autenticação novamente e imprime a mensagem acima no console.

Teste de disparidade entre campos de nova senha:

Este teste foi feito com o segundo link da Figura 14, que leva a “dir1/dir11/formulario.html” e, trata do caso em que as senhas colocadas em cada um dos campos de nova senha não batem, tem-se a seguinte resposta do servidor:



**Um erro foi identificado: Os campos de nova senha não batem, tente novamente!**

Figura 53: resposta a uma requisição POST de formulário mal sucedida por senhas diferentes nos campos de nova senha

Teste de falha de autenticação em tentativa de mudança de senha (erro de credenciais):

Do mesmo modo como quando o usuário tenta se autenticar usando uma senha antiga após tê-la alterado, o usuário que tentar alterar sua senha escrevendo suas credenciais atuais incorretamente receberá a mensagem da Figura 52 no console e terá a página de formulário reenviada para re-preenchimento no caso de ainda se ter interesse em mudar a senha. Pode ser visto abaixo:

Figura 54: Formulário preenchido antes da tentativa de alteração

ório de Micro eX Atualização de senha do usuá... G\_EA872K\_2023S2: Calend... +

calhost:3133/dir1/dir11/formulario.html

## EA872 - Laboratório de Programação de Software Bás

### Formulário para atualizar senha do usuário

Digite seu nome de usuário cadastrado:

Digite a senha atual:

Digite a nova senha (mínimo 8 caracteres):

Confirme a nova senha:

Figura 55: Página após tentativa de alteração com credenciais incorretas

Observando as duas imagens acima, é possível perceber que apenas é reenviada a página do formulário com os dados mantidos no caso de as credenciais atuais serem preenchidas incorretamente.

## O problema da implementação do parser reentrante:

Durante todo o processo de desenvolvimento do servidor durante a disciplina, foi recorrente a necessidade de analisar o código à procura de problemas e, por vezes, isso foi uma tarefa bastante difícil. Em concordância com isso, na versão final do servidor desenvolvido, surgiu um problema muito complexo de ser resolvido. Este problema, certamente está relacionado com o fato de o parser, nesta última versão, ter sido configurado para funcionar de maneira reentrante. Isso porque, na versão imediatamente anterior, cuja única diferença é o parser não reentrante, este problema ocorria, mas foi resolvido com facilidade após um tempo de análise do código e, ao tentar implementar a mesma solução para a nova versão, o problema persistiu.

Cabe explicar, então, o problema em questão. Ocorre que a nova versão do servidor, que tem parser reentrante, sofre uma queda por conta de erro “segmentation fault” quando estressada. Por várias horas o problema foi perseguido por meio da impressão de mensagens usando a função “printf()” para tentar isolar o problema e encontrá-lo. Todavia, infelizmente o problema não foi encontrado, o que significa que o servidor com região crítica reduzida pelo uso do parser reentrante não é completamente funcional, embora exerça muito bem sua função para um uso não exagerado.

Apresenta-se ainda, a solução encontrada para o problema de “segmentation fault” encontrado ao estressar o servidor com parser normal, não reentrante, citada anteriormente.

Após um período de depuração, notou-se no código uma diferença extremamente grande entre o número de chamadas “open()” e “close()”. Desse modo, ao analisar mais profundamente, foram encontradas várias ocorrências de “file-descriptors” sendo inicializados e nunca encerrados. Assim, embora tenha sido razoavelmente demorado encontrar o problema, sua solução foi bem simples, uma vez que foi um erro bastante grotesco. Portanto, bastou que fossem fechados esses “file-descriptors” no código para que o problema de “segmentation fault” não ocorresse mais.

Por fim, ao se tentar implementar a mesma solução para o servidor com parser reentrante, o problema persistiu, o que indica que deve estar nos blocos de código alterados para a adaptação ao parser reentrante ou nos blocos que tiveram comportamento afetado por isso. Nesse sentido, fez-se uma busca intensa em tais blocos citados, mas uma grande dificuldade se apresentou. Por causa de o servidor ter seu funcionamento dependente de mais de uma thread, ao imprimir mensagens para encontrar o problema, elas eram impressas mais de uma vez. Por esse motivo, como tentativa de circundar esse problema, as mensagens foram incrementadas com o id das threads, por meio da função “pthread\_self()”. Assim, foi minimamente facilitada a compreensão do que estava acontecendo com o servidor por meio do console. Contudo, apesar de todo o tempo e esforço despendidos a fim de resolver tal questão, o problema parecia incompreensível e, por conta do prazo, sua solução, infelizmente, teve de ser abandonada. Por fim, vale ressaltar que, embora uma solução não tenha sido encontrada, a incrementação do código com chamadas “close()” foi eficaz e o servidor, após isso, passou a sustentar mais tempo sendo estressado antes de ser derrubado por falha de segmentação.

Observação: como foi destacado no texto, esse problema afeta somente a implementação com parser reentrante, enquanto que **a versão com região crítica maior (parser não reentrante) não é afetada pelo problema** e pode ser estressada o quanto for desejado!

## **O problema de limite de requisições em série (resolvido!):**

Após a primeira entrega, foi notado o problema de threads sendo limitadas não apenas para requisições feitas em paralelo, mas também para requisições feitas em série. Por isso, foi procurado pelo problema, partindo-se do princípio que estaria relacionado à lógica de decremento e incremento da variável “cont\_threads” responsável por armazenar o número de threads em execução no momento. No entanto, com o uso do comando “ps” com diferentes opções, percebeu-se que o problema estava, na verdade, relacionado ao não fechamento de algumas threads que eram abertas. Isso se deu, por causa de um problema na lógica de espera por dados a serem lidos do soquete, mas foi rapidamente arrumada essa lógica e, o problema foi resolvido.

## **Limitações da implementação feita:**

Falando a respeito das duas implementações (com parser reentrante e sem), vale destacar o fato de que ambas não apresentam um número tão alto quanto esperado de requisições respondidas por segundo quando analisadas por meio de http-load. No entanto, comparando com os resultados obtidos na atividade cujo enfoque era estritamente este, é possível presumir que talvez o problema maior seja a máquina usada, uma vez que, em média, nos testes mais recentes, obteve-se 60 respostas/segundo enquanto que, nos testes antigos, a média era maior. Porém, de qualquer forma, não é possível afirmar isso com certeza, uma vez que os testes antigos foram feitos com menos tempo de análise e a versão do programa era outra, logo, a comparação é um tanto injusta. Por este motivo, para todos os efeitos, destaca-se aqui que isso é uma limitação da implementação desenvolvida, o número máximo de requisições respondidas por segundo é baixo.

Ademais, ressalta-se o problema de “segmentation fault” que ocorre após um tempo indeterminado de estresse do servidor com parser reentrante. Este problema, embora não ocorra na versão do servidor anterior a implementação do modo reentrante tanto no analisador léxico como no sintático, ocorre com a versão em que isso está implementado e, a solução deste problema se mostrou bastante complexa. Isso gera uma limitação na estabilidade do servidor, já que não é possível prever seu tempo até a queda num caso de estresse. Logo, o servidor corre risco de ser encerrado a qualquer momento em uma situação de congestionamento. No entanto, foram feitos testes e melhorias com relação a esta implementação (reentrante) e, embora ainda apresente tal problema, o tempo que o servidor leva até ser derrubado está maior, como foi explicado anteriormente. Desse modo, apesar da instabilidade, é possível afirmar que ele poderia ser utilizado em situações normais de uso, nas quais não fosse exigida uma atividade excessiva do servidor.

## **Trabalhos futuros de melhoria do sistema que poderão ser feitos:**

Primeiramente, é importante colocar o problema de “segmentation fault” do servidor de parser reentrante, discutido anteriormente, em destaque, uma vez que essa é a limitação mais evidente do servidor. Isso porque, em um servidor aberto a uso público, como o de um site de vendas ou o de uma rede social, por exemplo, esse problema afetaria gravemente seu funcionamento, pois esses sites, principalmente o segundo citado, sofrem constantemente com estresse. Logo, para estes casos de exemplo, não seria nada interessante ter um servidor instável que pode ser derrubado a qualquer momento em uma situação de estresse contínuo. Portanto, uma melhoria bastante evidente e significativa que poderá ser buscada no futuro é a solução do problema de queda do servidor com parser reentrante por estresse. Porém, para o servidor com parser não reentrante, isso não é um problema e, conseqüentemente, sua solução não seria uma melhoria a ser feita.

Ademais, em segundo lugar, uma melhoria bastante expressiva, seria aumentar o número de requisições respondidas por segundo pelo servidor, já que este número se mostrou baixo. De qualquer forma, este problema poderia ser resolvido com uma depuração

aprofundada do código em busca de possíveis otimizações que trouxessem um resultado favorável à agilidade do servidor em responder requisições. Para tanto, seria possível revisar as regiões críticas protegidas por “mutex”, buscando reduzi-las, principalmente no servidor com parser não reentrante, no qual o número de instruções dentro da região crítica é muito maior (“yyparse()” dentro de região crítica).

Em suma, o que cabe de melhoria ao servidor seria a implementação do modo reentrante do parser de forma que não houvesse problema de segmentação e, somado a isso, seria interessantíssima uma revisão na lógica por trás da implementação e, talvez, nas funções utilizadas. Isso porque, pode ser possível substituir funções e chamadas feitas de modo a agilizar o processo de resposta a requisições, melhorando, por conseguinte, o funcionamento do servidor e o número de respostas a requisições por segundo.

## **Comentários, críticas e sugestões a respeito das atividades:**

Esta disciplina, foi bastante proveitosa para aprendermos a respeito de chamadas de sistema, processos, threads e outros tópicos relacionados a sistemas operacionais e software. Desse modo, pude expandir meus conhecimentos relacionados à computação de uma maneira muito boa. Logo, agradeço pela disciplina e pelo empenho do professor. No entanto, surgiu-me uma única crítica a fazer. Tal crítica se dá a respeito do feedback das atividades desenvolvidas durante o semestre. Isso porque, seria muito bom que tivéssemos um feedback constante durante a disciplina para termos certeza de que estamos entendendo a matéria e de que, no caso dos últimos laboratórios, nosso código não está acumulando problemas para serem resolvidos.

No mais, deixo meus agradecimentos ao professor e a todos os envolvidos na confecção dos roteiros que usamos para desenvolver as atividades da disciplina.

Observação: Embora não me pareça apropriado usar primeira pessoa em um relatório como este, usei nesta última seção para conseguir expressar de maneira mais clara minha opinião a respeito da disciplina.