Universidade de Brasília

Departamento de Ciência da Computação

Teleinformática e Redes 2 - Turma A – 2018/1

Professor: João Gondim

Alex Siqueira Lacerda 16/0047692

Jonatas Gomes Pinto Junior 14/0146407

**Um Inspetor HTTP baseado em Proxy Server**

Objetivo

O objetivo desse trabalho é desenvolver um inspetor HTTP baseado em Proxy Server, aplicando o conteúdo aprendido em sala de aula sobre camada de aplicação e transporte. O servidor deverá receber requisições HTTP processando seu conteúdo de cabeçalho para verificação do endereço de destino. Deve ser utilizado um navegador web, devidamente configurado para o uso do proxy. O sistema também deve implementar duas outras funcionalidades, um Spider e um cliente recursivo. O Spider gera uma arvore hipertextual com as URLs obtidas do objeto e o cliente recursivo salva o conteúdo de uma URL.

Introdução

Ao estudarmos a camada de aplicação e os protocolos de rede que existem, dois protocolos muito importantes não podem ser ignorados, dado a abrangente frequência de utilização deles na internet, esta que é usada por todos. São os protocolos HTTP, que é usado na camada de aplicação, e o protocolo TCP, que é utilizado junto ao protocolo HTTP, na camada de transporte. Uma comunicação feita entre um browser cliente e um servidor que armazena as páginas de internet será realizada com estes protocolos, e, então, estuda-los é essencial para que seja possível compreender e criar qualquer aplicação em uma rede que propõe analisar e armazenar páginas que estão disponíveis na internet, uma aplicação de exemplo é o caso de um servidor proxy.

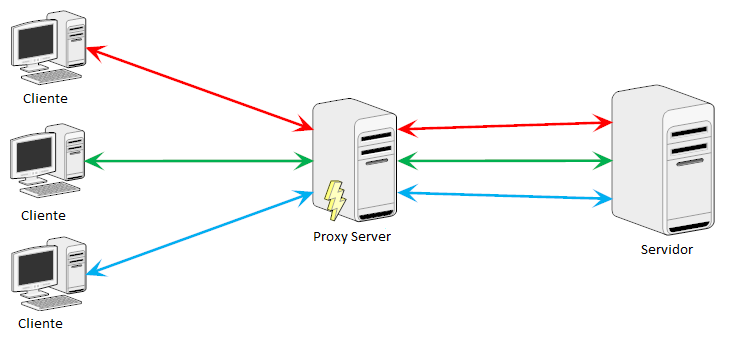
[](https://github.com/ArashPartow/proxy)Proxy Server Web:

Figura https://github.com/ArashPartow/proxy

Um dos objetivos desse trabalho é a implementação de um servidor proxy, mas o que é um servidor proxy? Um servidor proxy, que também pode ser chamado de **cache Web**, é uma funcionalidade da rede que recebe requisições HTTP de um servidor Web. O proxy tem sua própria memória, e o mantém, junto dele, copias dos objetos que foram requisitados. O browser de um usuário pode ser configurado de uma maneira que suas requisições HTTP sejam direcionadas primeiro ao proxy, uma vez feito isso, cada uma das requisições de um objeto que o browser faz é direcionada primeiro ao proxy server.

Funcionamento:

*1. O browser faz uma conexão com o servidor proxy e envia a ele uma requisição HTTP para o objeto.*

*2. O servidor proxy realiza uma verificação se já existe uma cópia do objeto armazenada localmente. Se existir, envia o objeto ao browser do cliente, dentro de um response HTTP.*

*3. Caso o objeto não exista, o servidor proxy abre uma conexão com servidor de origem, enviando uma requisição HTTP do objeto para conexão. Recebendo, o servidor de origem envia o objeto ao servidor proxy, dentro de uma resposta HTTP.*

*4. O servidor proxy, armazena o objeto localmente e envia uma cópia dentro de uma mensagem de resposta HTPP, ao browser do cliente (Pela conexão já estabelecida entre o browser do cliente e o servidor proxy).*

Um exemplo de utilização de um proxy é que ele pode ser filtrar e bloquear requisições enviadas. É possível definir que toda requisição de uma máquina deve passar por um proxy local. Esse, por sua vez, impede que requisições HTTP, que sejam destinados a um host especifico cheguem ao servidor.

Protocolo TCP

O protocolo TCP, é um protocolo da camada de transporte, usado pelo protocolo HTTP para enviar suas mensagens. O TCP é ideal para escolha de uso junto ao HTTP diante do fato que ele é um protocolo que garante transferência confiável de dados, garante que a entrega de todos os pacotes irá ocorrer em ordem e não vai sofrer com perdas. Para uma aplicação que tem o propósito de enviar e receber objetos, é fundamental garantir a entrega total da mensagem.

O protocolo TCP é um protocolo orientado a conexão, um protocolo que faz os processos se comunicarem realizando handshake (troca de mensagens para estabelecer apenas a conexão)

Uma das funções mais importantes do protocolo TCP é realizar a multiplexação e demultiplexação de mensagens sendo enviadas para ou por processos executando em um hospedeiro. A cama de transporte tem como principal função realizar a comunicação logica entre as aplicações, ou seja, realizar a comunicação entre processos executando em hospedeiros diferentes de tal forma que tal comunicação nunca seja de conhecimento da aplicação.

O protocolo TCP também possui um controle de congestionamento, sendo um serviço dirigido a internet como um todo, o controle de congestionamento do TCP, impede que qualquer outra conexão TCP atrapalhe os enlaces e roteadores entre os hospedeiros que estão com uma quantidade excessiva de tráfego. O TCP permite igualdade de banda entre as conexões daquele enlace, feito pela regulagem da taxa de envio de trafego do remetente.

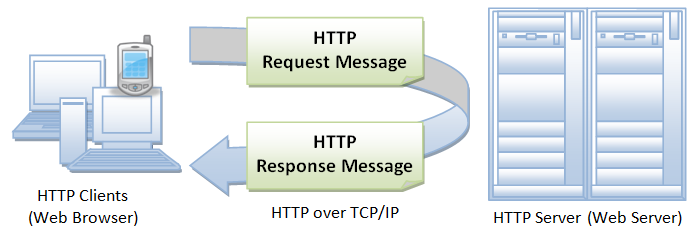
[](http://www.ntu.edu.sg/home/ehchua/programming/webprogramming/http_basics.html)Protocolo HTTP

Figura 2 http://www.ntu.edu.sg/home/ehchua/programming/webprogramming/http\_basics.html

HTTP é a uma sigla usada para representar o protocolo que existe na camada de aplicação que é chamado de *Hypertext Transfer Protocol*. Protocolo de comunicação utilizado para realizar a comunicação entre servidor e cliente conectados na WEB, eles conversam entre si por meio da troca de mensagens HTTP. O Protocolo HTTP define o funcionamento dessas mensagens e o modo de como estas são trocadas.

O HTTP define como os clientes requisitam as páginas aos servidores e como estas são transferidas para o cliente utilizando protocolo da camada de transporte TCP.

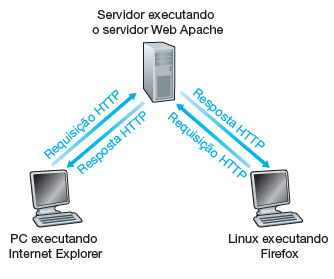
Comportamento de requisição HTTP

Figura redes de computadores e a internet KUROSE

O cliente primeiro inicia uma conexão TCP com o servidor, com a conexão estabelecida, os processos entre o browser e o do servidor acessam o TCP por meio de suas interfaces *socket*. No lado do cliente essa interface *socket* é a porta do processo cliente e a conexão TCP, do lado do servidor ela é a porta entre o processo servidor e a conexão TCP. O cliente envia para o servidor mensagens de requisição (request HTTP) para interface *socke*t, que responde para o cliente (response HTTP) que recebe em sua interface *socket*. De uma maneira semelhante, o servidor HTTP recebe mensagens de requisição de sua interface e envia mensagens de response para sua interface. Assim que o cliente envia uma mensagem, a mensagem sai de suas mãos e passa para a do TCP, que é responsável por entregar a mensagem ao destino servidor de maneira intacta. Do mesmo modo que toda mensagem no processo servidor também chegará intacta ao cliente. Podemos ver nesse ponto as vantagens de se possuir uma arquitetura baseada em camadas. Ao utilizar o protocolo TCP da camada de transporte, o HTTP não precisa se preocupar com dados perdidos ou com detalhes de como o TCP se recupera da perda de dados ou os organiza dentro da rede.

O protocolo HTTP é denominado um protocolo sem estado, pois não mantém informação alguma sobre os clientes, caso um determinado cliente solicita o mesmo objeto duas vezes em poucos segundos, o servidor não avisa que acabou de enviar, ele simplesmente manda de novo o objeto, pois já se esqueceu complemente o que havia feito antes.

O HTTP possui várias mensagens de requisição, onde a mais comum envolve o método GET, utilizado para recuperar o conteúdo de uma página web ou de um de seus objetos.

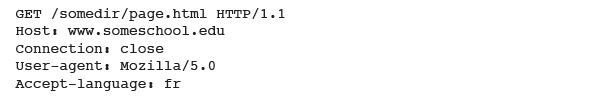
Exemplo de mensagem de requisição HTTP:

Figura : redes de computadores e a internet KUROSE

Vamos entender esta requisição, a primeira linha está pedindo (GET) que o servidor HTTP envie o objeto **/somedir/page.html**, um documento escrito em html, e logo após temos a versão HTTP que está sendo utilizada nesta comunicação **HTTP/1.1.** na linha seguinte, está descrito o nome do servidor que está hospedando esse objeto requisitado,[**www.someschool.edu**](http://www.someschool.edu), que evidentemente será traduzida para o endereço IP que o representa. Logo após na linha 3, está definido no campo que a conexão realizada para recuperação não será persistente. A próxima linha, define o campo de cabeçalho, que diz qual o tipo do agente que está realizando a requisição, um navegador **Mozilla/5.0**, e a ultima linha, temos um campo que define que se a página possuir uma versão em francês está versão é que deve ser enviada preferencialmente.

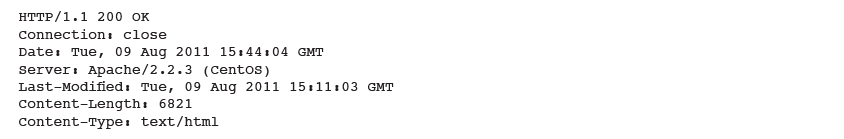
Exemplo de mensagem de resposta HTTP:

Figura redes de computadores e a internet KUROSE

Nesta imagem perceba que a mensagem de resposta possui um formato semelhante à de requisição: contendo o estado da resposta enviada pelo servidor, indicando se a requisição foi aceita corretamente. Temos a versão do protocolo que foi utilizada novamente e o código de status com mensagem correspondente a tal código (200 OK). Temos diversos campos de cabeçalho seguidos de seus respectivos atributos, e finalmente o **payload**, da mensagem que, para o exemplo, irá conter o objeto requisitado.

Arquitetura desenvolvida.

Para realização deste trabalho, foi escolhido para implementação a linguagem C++. Os sistemas implementados foram: Servidor Proxy, Spider, Cliente recursivo (dump) e um manipulador de string funcionalidade extra que facilitou a implementação das funcionalidades.

Servidor Proxy

Servidor proxy simples que age sobre a arquitetura cliente e servidor, o proxy implementado, portanto, pode receber diversas requisições HTTP vindas de diversos clientes que foram configurados para utiliza-lo. Estas requisições são filtradas pelo proxy e após isso repassadas para seus devidos servidores HTTP pelo próprio proxy, que neste caso após receber a requisição age como cliente na hora de verificar se a requisição pode ser atendida, assim podendo retornar o objeto para o cliente que solicitou a requisição.



Spider e Cliente recursivo

O Spider é uma funcionalidade feita para enumerar as URLs contidas em um site formando uma árvore hipertextual.

A funcionalidade cliente recursivo faz um dump de todo conteúdo de um site a partir de um URL específica e salva.

HTTP\_Request

HTTP\_Response

Extra: Foi implementado um manipulador de strings, como facilitador.

Manipulador de String – Facilita o parser e a manipulação nos códigos.

Como executar o programa

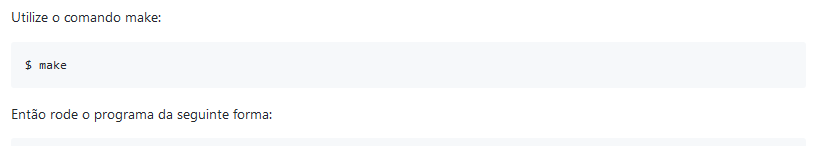


Figura No terminal utilize os comandos acima

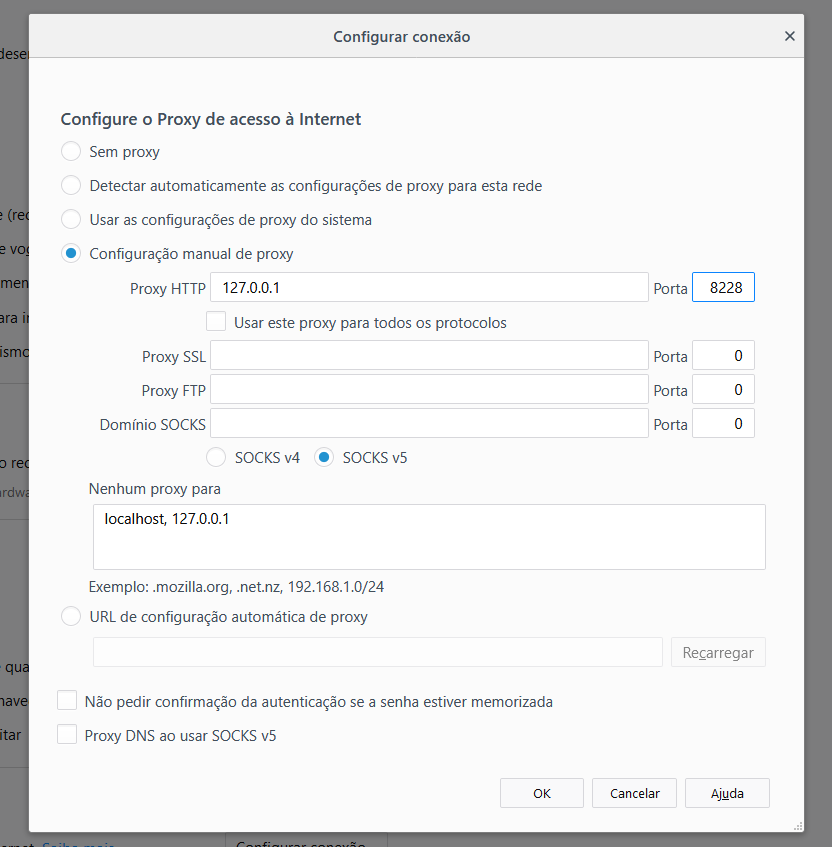
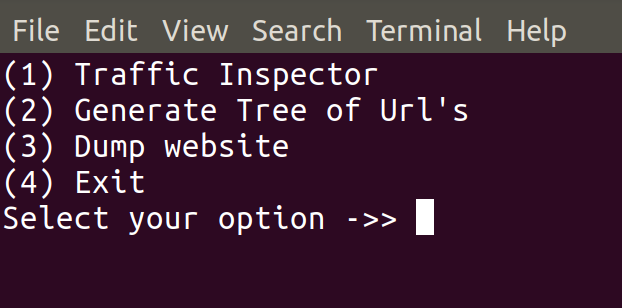
Problemas durante implementação

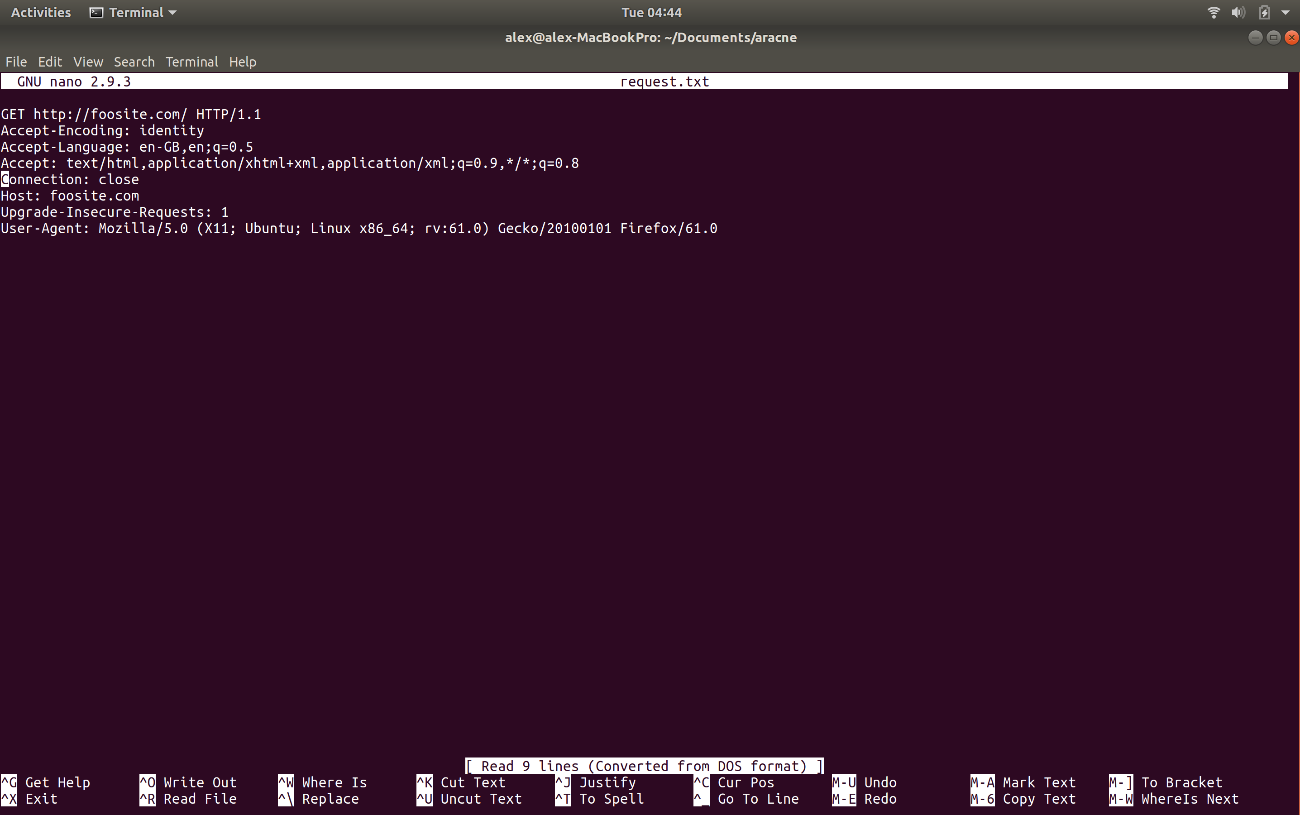
Programação baixo nível em sockets torna o funcionamento do programa instável por existirem muitos detalhes a serem considerados. (Nem sempre funciona corretamente) -Complexidade de fazer o parse HTTP e HTML, funciona pra maioria dos casos, mais existem muitas situações em que se comporta mal. -Não conseguimos criar a interface gráfica devido à grande quantidade de problemas encontrados na lógica do programa. -Muitas funções blocantes que exigem conhecimento de Programação Concorrente para implementação de Threads (não foi feito).

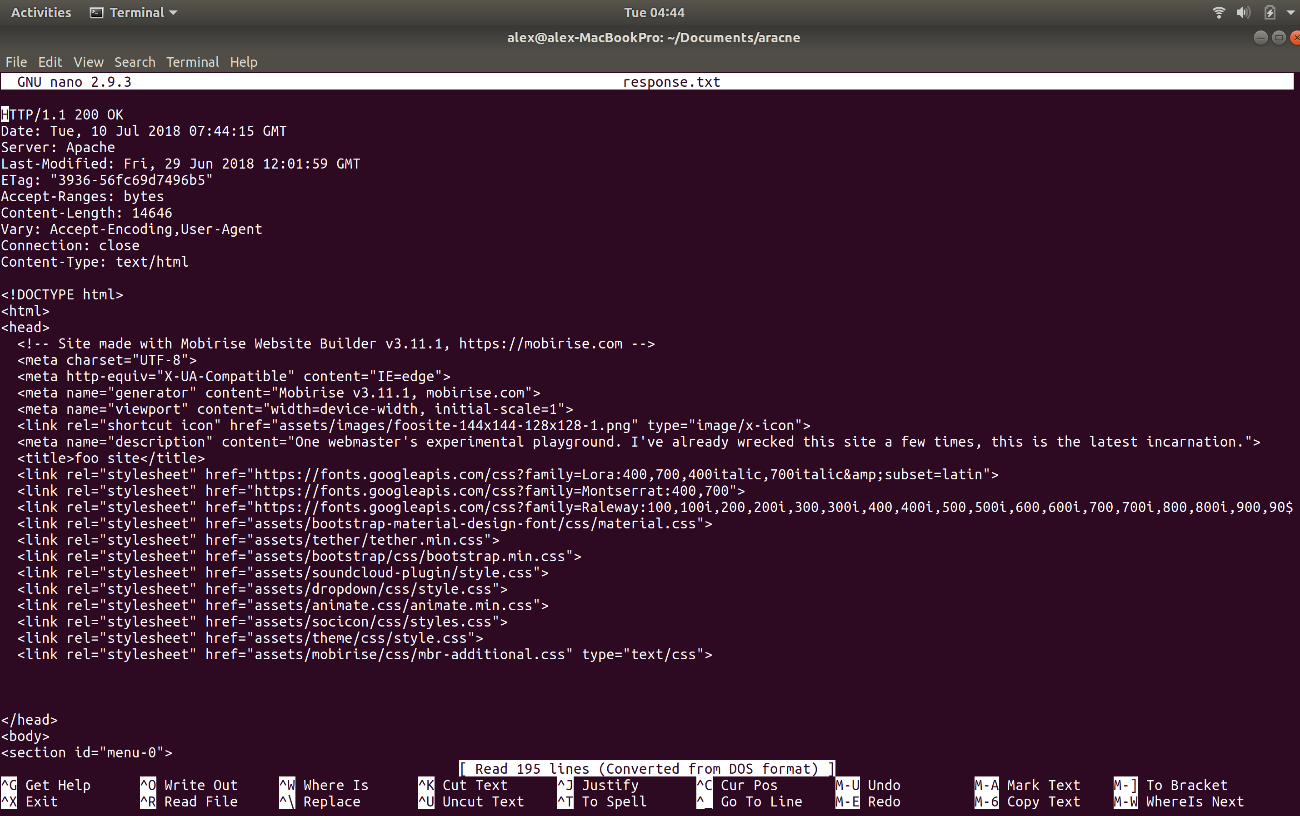
Documentação do código desenvolvido

Código documentado encontrasse no GitHub:

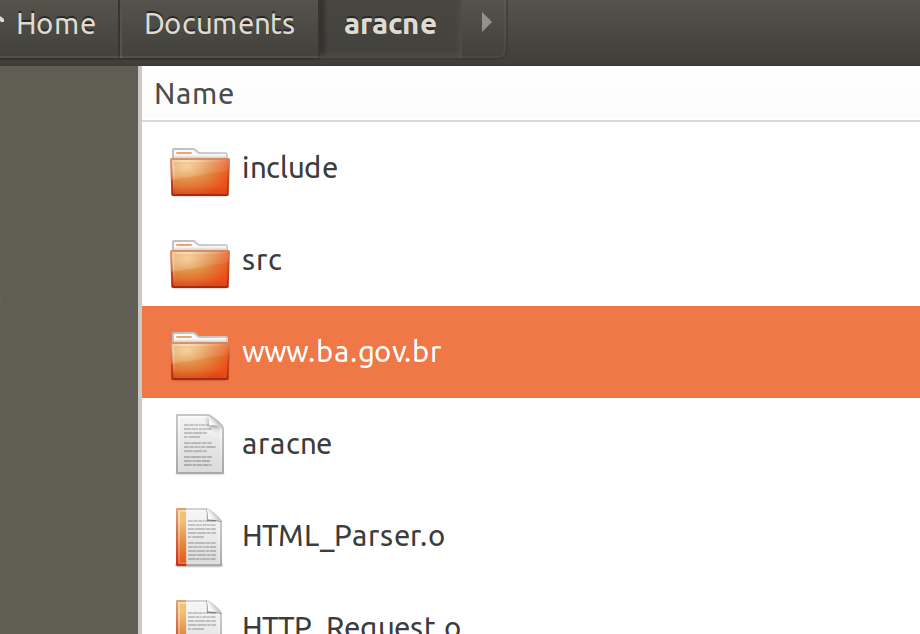
<https://github.com/siqueiralex/aracne>

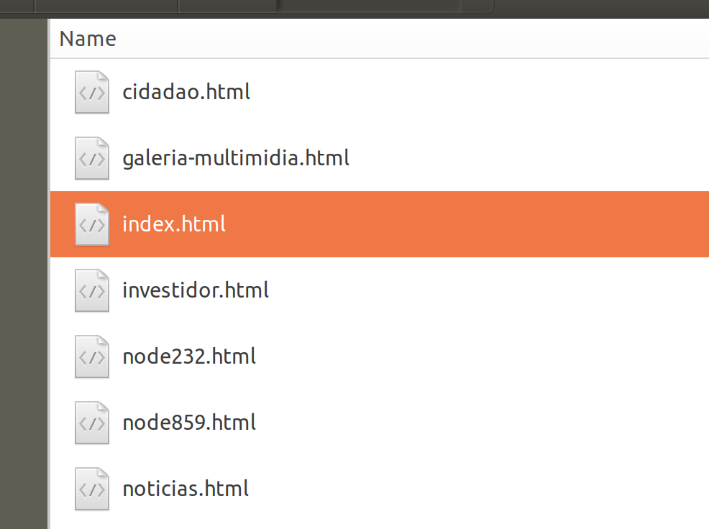
Funcionamento do programa

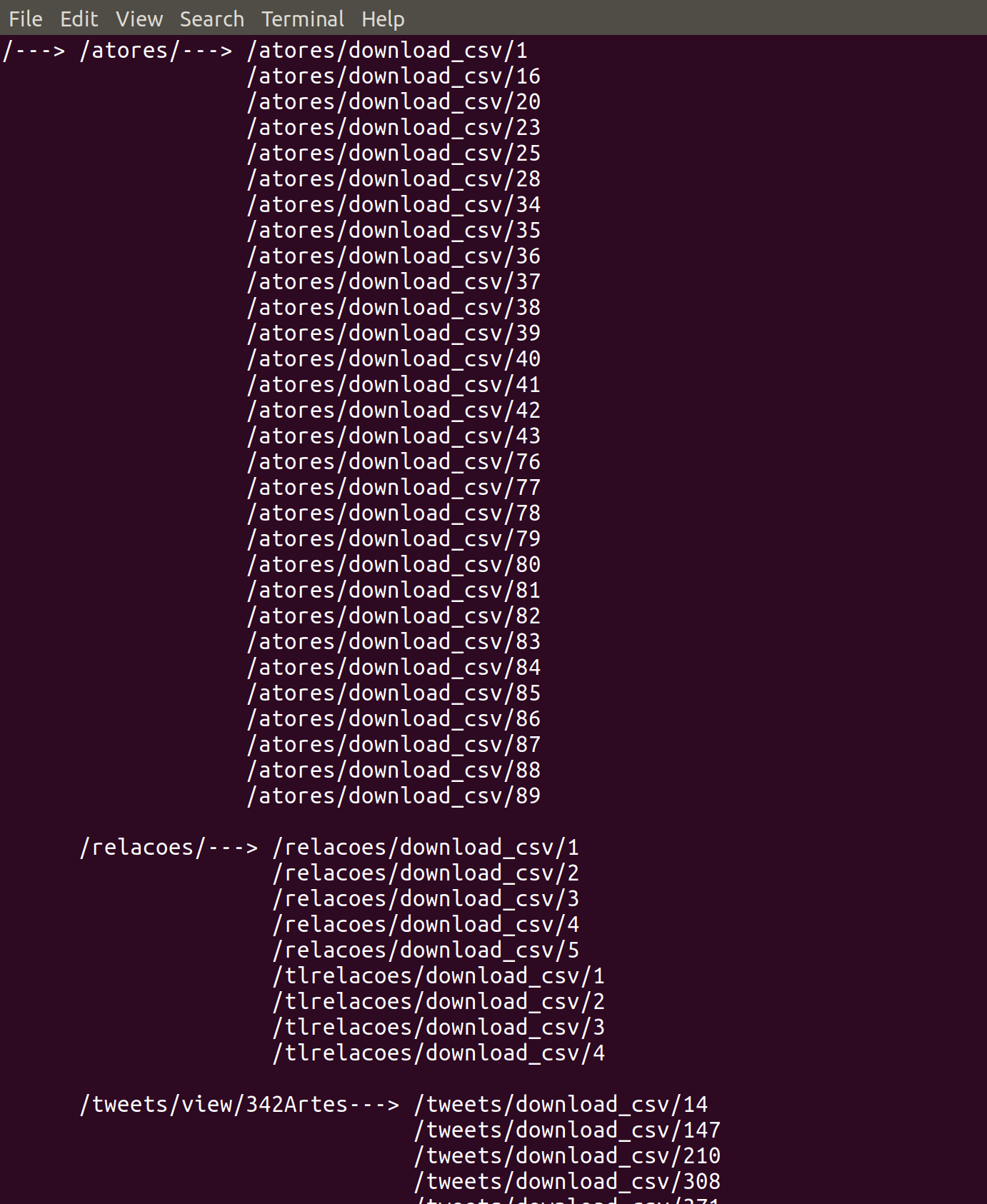


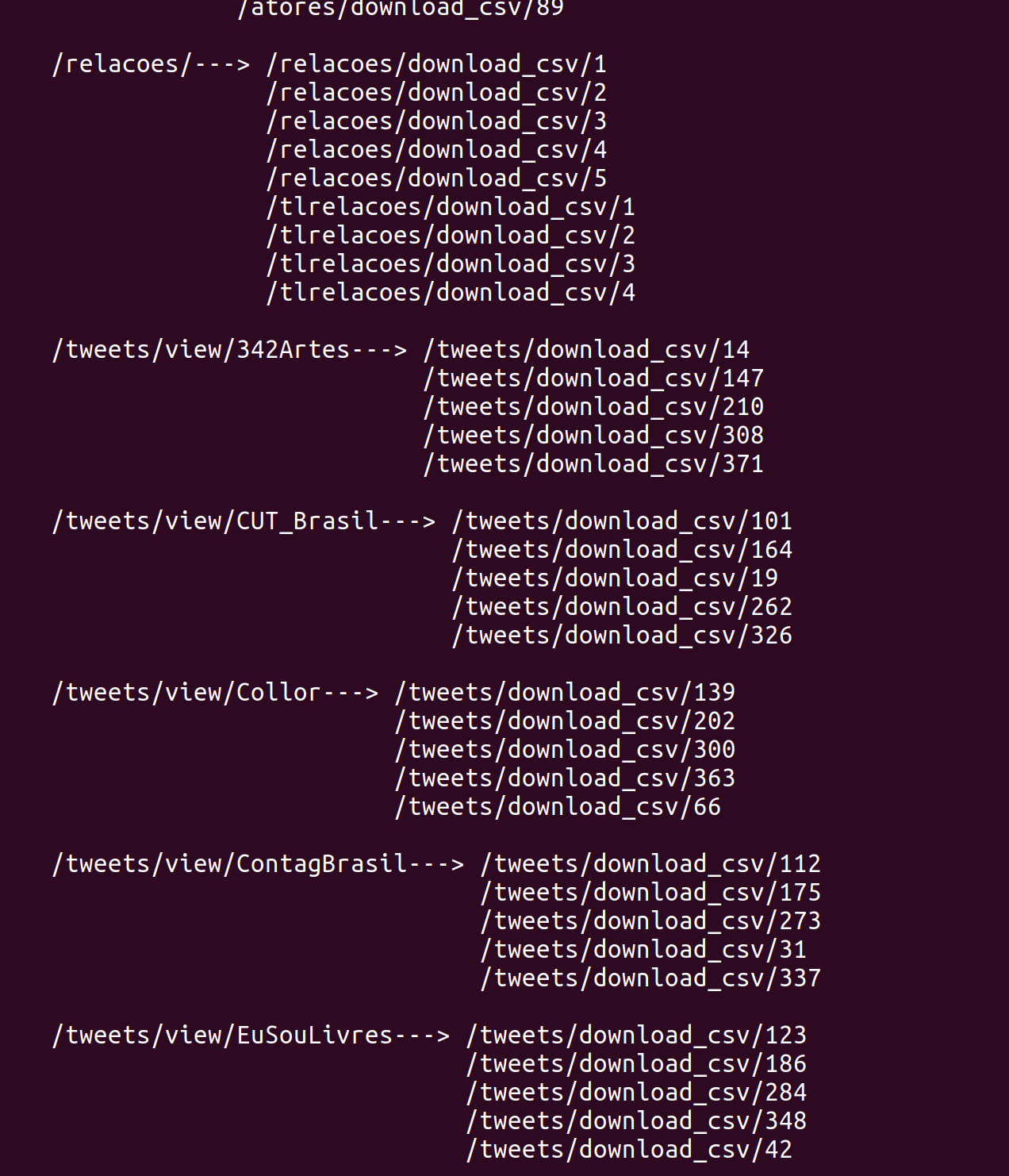












Conclusão

A realização do trabalho facilitou o entendimento de muitos conceitos vistos em sala de aula, visto que muito da teoria envolvida pode ser visto de forma mais clara após a realização da aplicação do conteúdo e exemplos práticos.

Neste trabalho é implementado um servidor proxy simples, apesar de não ser o mais correto que se pode ter, abre portas para entendimentos mais complexos em futuras implementações das aplicações em rede. O trabalho exigiu o estudo e a compressão dos protocolos que foram definidos e a interação com um cliente já implementado. A implementação dos módulos capazes de interagir com outras aplicações e programas diferentes não é uma tarefa trivial, mas é uma tarefa necessária e que exige ser praticada quando é se falado de aplicações em rede, o cliente e o servidor nem sempre são desenvolvidos no mesmo processo mas isso não deve ser um impedimento para que eles se comuniquem. Descobrir o funcionamento dos protocolos e suas funcionalidades foram de suma importância para esse trabalho.

Referências

KUROSE, James F; ROSS, Keith W. Transport Layer. In: **Computer Networking – A Top-Down**

**Approach**. Sexta edição. Estados Unidos da América: Pearson Education, 2013. Cap. 3, p. 185-283.

KUROSE, James F; ROSS, Keith W. Application Layer. In: **Computer Networking – A Top-Down**

**Approach**. Sexta edição. Estados Unidos da América: Pearson Education, 2013. Cap. 2, p. 83-184.