Implementação de um servidor *Proxy*

Filipe Teixeira - 14/0139486
Lucas Santos - 14/0151010
Marcos Tonin - 14/0153233
Victor Fabre - 15/0022948

***Resumo*—Este documento apresenta a implementação de um servidor *proxy* por parte do grupo, explanando conceitos teóricos relacionados com o servidor *proxy*, protocolo HTTP e protocolo TCP. O documento também descreve a arquitetura do sistema desenvolvido, incluindo a documentação do código construído pelos membros do grupo, feita por meio do *doxygen*. E finalmente a última sessão, evidencia os resultados obtidos pelo sistema desenvolvido.**

I. Introdução Teórica

A. Proxy Server Web

Um *proxy server* é um servidor que age como um intermediário entre o cliente e o servidor, assim podemos dizer que para servidor o *proxy server* é um cliente, e para o cliente o *proxy server* é um servidor.

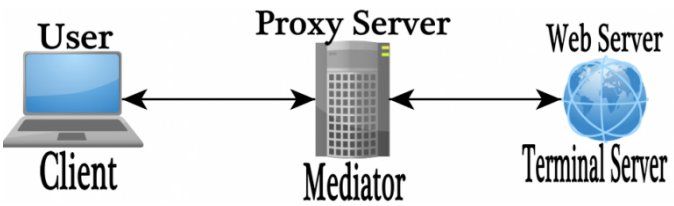

Figura 1. Representação do *Proxy*

Mas por essa simples definição não se nota qual a vantagem do *proxy* ou até mesmo sua utilidade, isto é verdade se tratarmos apenas da da conexão entre o *proxy* e o servidor, porém a conexão feita entre o cliente e *proxy* muda e melhora, pois o cliente manda sua requisição ao *proxy server* e este tem o trabalho de verificar a solicitação e facilitar a mesma quando for possível, por exemplo :

- A utilização de cache, armazenando dados.
- Controlar/filtrar o que se está requisitando.
- Pode alterar as requisições feitas pelo cliente ou a resposta do servidor.

O tipo de proxy mais utilizados hoje em dia é *Proxy Server Web* que provê todas as utilidades de um proxy ao acesso a internet, mas especificamente ao conteúdo na *World Wide Web*, por exemplo, se um cliente pede dados de uma fonte da Internet, esta requisição sai do *browser* vai primeiramente no *Proxy Web Server*, aonde é verificado se o proxy tem o dado que o cliente procura ou se pode acessar o local deste dado, se sim o proxy comunica com servidor web e transmite de volta ao cliente. Alguns exemplos de *Proxy Server Web* : **PHProxy, Zelune, Glype, PHPWebProxy, CGIProxy**.

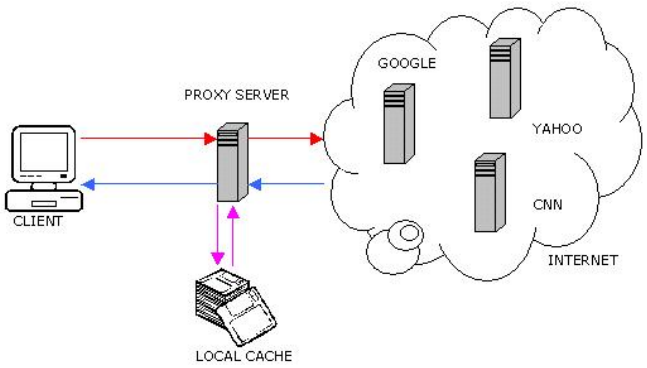

Figura 2. Representação do *Proxy com funcionalidade da cache*

B. Protocolo TCP

Os dois protocolos mais importantes da camada de transporte são UDPUser Datagram Protocol e TCP (*Transmission Control Protocol*). O protocolo de transporte fim-a-fim TCP tem por característica entrega ordenada de pacotes e confiabilidade na entrega - sem perda de pacotes, sem erros e sem duplicação- logo este protocolo permite uma comunicação mais segura do que o protocolo UDP que não garante entrega ordenada nem a entrega de todos os pacotes. Outra diferença em relação ao UDP é que o TCP é orientado a conexão, tendo um ”pré-passo”que é o handshaking que basicamente o cliente ”conhece”o servidor através de mensagens de controle.

A confiabilidade do protocolo do modelo cliente-servidor TCP é garantida utilizando a noção de número de sequência e o cálculo de *checksum* para os segmentos. Para cada segmento de dado que chega, o sistema receptor retorna um ”reconhecimento”do segmento que chegou, outro mecanismo usado é *timeout* que quando for estourado, o segmento em que foi acionado (vale salientar que para cada segmento se aciona um *timeout*) será reenviado, caso seja duplicado o servidor saberá devido ao número de sequência.

C. Protocolo HTTP

Protocolo que atua na camada de aplicação, também conhecida como protocolo de comunicação, vem do nome *Hypertext Transfer Protocol* que é protocolo mais utilizado na Internet, seu objetivo é a transferência de arquivos.

O funcionamento do HTTP é basicamente uma conexão entre navegador e servidor, que ocorre com o navegador efetuando um pedido HTTP, que é tratada pelo servidor e depois ocorre uma resposta HTTP.

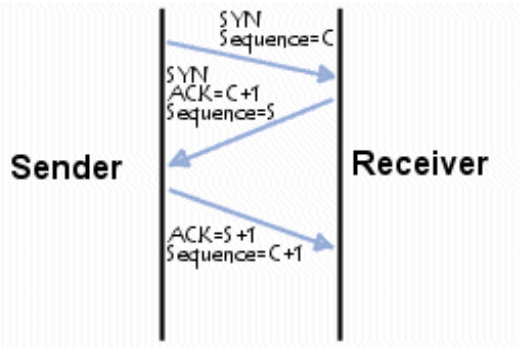

Figura 3. Representação do *Proxy com funcionalidade da cache*

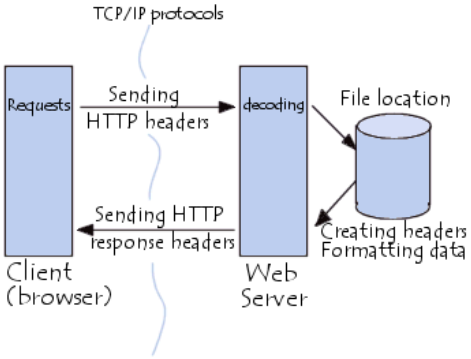

Figura 4. Representação do *Funcionamento HTTP*

II. Arquitetura do Sistema Desenvolvido

1) Apresentação: O objetivo do trabalho era fazer um servidor proxy em C que recebe-se os dados enviados pelas requisições feitas via browser de um navegador, utilizando uma porta especifica (maior que 1023) e com esses dados fazer parse dos campos HTTP necessário para enviar ao servidor interno da Internet, com os campos é feito a filtragem e também "cacheamento"de uma página se for possível e se a mesma não estiver na cache já, também é possível examinação do cabeçalho e também podendo modificar o cabeçalho.

2) Explicação: A função básica do proxy é responsável por receber dados das requisições feitas no browser, fazer o parse dos valores das requisições além de fazer todos os passos para receber uma requisição (socket -¿ bind -¿listen -¿ accept), fazer o parser escrever a requisição para mandar para o servidor interno da Internet e depois receber os dados.

- **Cache**: Depois de feito a verificação da URL e supondo que está tudo certo, analisa se está requisição já está na cache e se a página original não foi alterada, caso não esteja na cache armazena-se os dados necessário para cache.
- **Inspeção cabeçalho**: Intercepta uma requisição HTTP e permite examinar o cabeçalho e também altera-lo .
- **Filtragem**: Verificar se a URL da requisição se encontra no arquivo *blacklist.txt*, sendo bloqueado ou se está URL está em *whitelist.txt* que tem a relação dos sites que são acessáveis sem maiores problemas. Feito isso é passado um "OK!"para a cache armazenar os dados e referentes. Por fim, para analisar *denyterms.txt* analisava os dados que o servidor web estava enviando, caso tivesse algum termo proibido, se cancelava o envio de dados da página.

3) Relação entre os Principais Componentes: Para fazer o trabalho foi definido quatro funcionalidades principais : Funcionamento básico, cache, filtragem e inspenção de cabeçalho. De modo que a inspenção de cabeçalho não se mistura com filtragem e cache, ou seja, para se fazer a inspeção e modificação de cabeçalho, não se utilizava os módulos de filtragem e cache. Para se fazer isso tem que se passar o argumento -I"por linha de comando na hora da execução do arquivo.

Os módulos de filtro, cache e funcionamento básico teve ser feito de modo que os três possam conversar entre si. O primeiro módulo utilizado é funcionamento básico que faz as operações necessárias para obter a URL, desde de socket-¿bind-¿listen até fazer o parser da requisição HTTP, com esta URL obtida é pesquisado para ver ser a URL está na *whitelist* ou na *blacklist*, se a mesma contem algum termo proibido.

Com o resultado desta pesquisa é possível decidir se continua (quanto a URL está na *whitelist* ou se não está em nenhum dos casos negativos) a operação com o cliente ou se envia ao usuário uma mensagem de erro, neste caso dizendo que aquele site é bloqueado -que é o caso da URL estar na *blacklist* ou conter na *whitelist*-.

Supondo que esteja tudo OK com a URL, começa o trabalho do módulo da cache, neste momento é verificado se a URL em questão está na cache e se ainda não expirou, caso esteja na cache é carregada imediatamente. Se não tiver continua as requisições para buscar os dados da página que quer se acessar. Neste ponto é verificado se tem termos proibidos nos dados da página, se tiver é enviada uma mensagem de erro, caso não tenha termos proibidos nos dados a cache armazena os dados da página e outros valores que lhe são importante.

III. Documentação do Código

A documentação do código foi gerada por meio do *doxygen* e se encontra disponível no apêndice ao final do relatório na seguinte ordem:

1) *estruturas.h*;
2) *cache.h*;
3) *inspecao.h*;
4) *filtragem.h*.

IV. Relação das Bibliotecas Utilizadas

A seguir a lista de bibliotecas usadas:

```
#include <netinet/tcp.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <ctype.h>
#include <setjmp.h>
#include <signal.h>
#include <sys/time.h>
```

```
10 #include <sys/types.h>
11 #include <sys/wait.h>
12 #include <sys/stat.h>
13 #include <fcntl.h>
14 #include <sys/mman.h>
15 #include <errno.h>
16 #include <math.h>
17 #include <pthread.h>
18 #include <semaphore.h>
19 #include <sys/socket.h>
20 #include <netdb.h>
21 #include <netinet/in.h>
22 #include <arpa/inet.h>
23 #include <time.h>
```

V. Resultados

Foram feitos testes no proxy desenvolvido para verificar seu comportamento. Os testes foram realizados utilizando-se do navegador Mozilla Firefox configurado para rodar utilizando um proxy local e não armazenar nenhuma memória cache do navegador.

Os alunos testaram alguns domínios que conseguiram ser carregados de maneira efetiva e em sua maioria correta, sendo eles:

- pudim.com.br
- xputer.de
- www.guimp.com
- www.zombo.com
- www.unb.br
- www.example.com

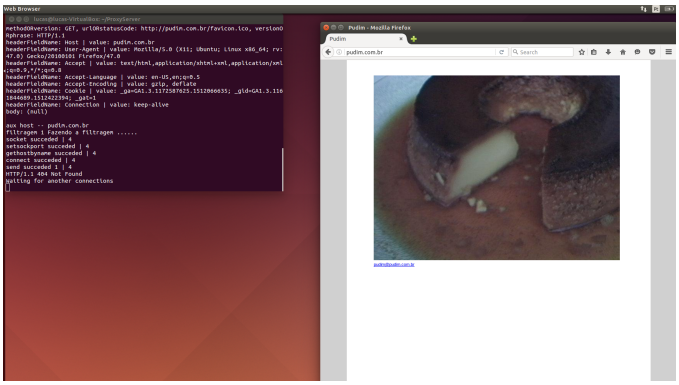

Figura 5. Observação do bom funcionamento do *proxy* para o carregamento do domínio http://pudim.com.br

Os alunos também realizaram testes para filtragem de conteúdo, onde o bloqueio de tais domínio foi bem-sucedido:

- www.youtube.com.br (filtrado - blacklist)
- www.netflix.com.br (filtrado - blacklist)

Testes também foram implementados no contexto da cache implementada, verificando o funcionamento da mesma como ilustrado na Figura 7

VI. Problemas Encontrados e Trabalhos Futuros

O funcionamento do *proxy* é variável, pois para alguns sites o carregamento se dá de maneira quase imediata, e para outros o carregamento demora minutos. Os alunos não foram capazes

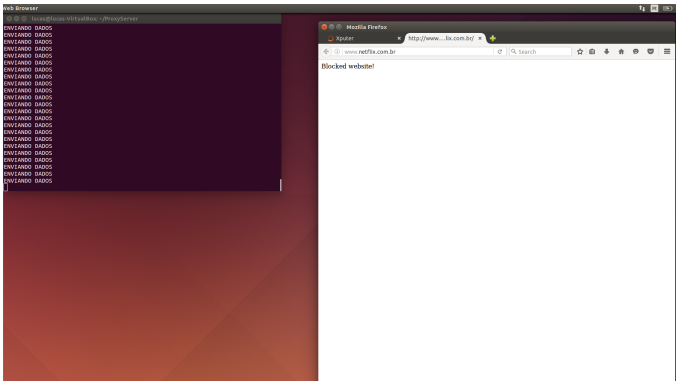

Figura 6. Verificação do funcionamento da filtragem implementada.

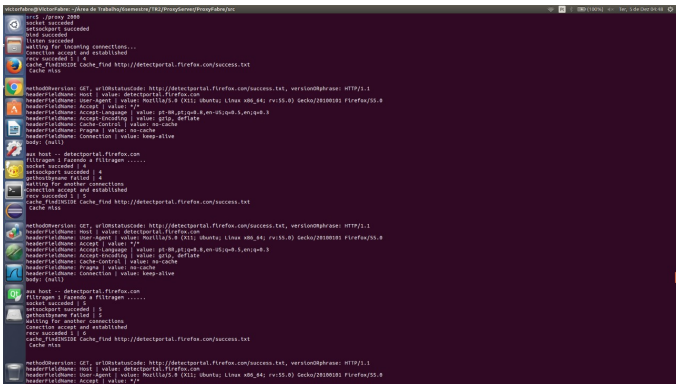

Figura 7. Ilustração da execução da cache, resultando em uma falta (*miss*).

de identificar o problema de implementação que causou tal comportamento por parte da *proxy*.

Outro problema percebido pelo grupo, é que a cache, da maneira que foi implementada, é apagada ao encerrar o programa.

A implementação da inspeção de código não apresenta uma janela de interface gráfica, e sim uma interface realizada pelo terminal por meio de *printfs* e *scanfs*.

VII. Instruções para Compilação e Execução do Código

Com o objetivo de facilitar a compilação dos módulos do projeto, foi feito uso de um arquivo *makefile*. Para compilar o código basta acessar o terminal a partir do diretório *"src"* e então executar o comando *"make"*. Para executar o programa basta inserir o seguinte comando no terminal aberto no diretório *"src"*: ./proxy (modo) (número da porta). Lembrando que é necessário configurar o navegador para rodar utilizando um proxy local na mesma porta inserida como argumento na linha de comando.

Existem três *flags* que podem ser utilizadas, apenas uma por vez, para o modo durante a execução:

- -i: modo de inspeção
- -f: modo de filtragem
- -c: modo de cache

Referências

[1] Proxy. Disponível em : https://pt.wikipedia.org/wiki/Proxy.

[2] Proxy Server. Disponível em :http://whatis.techtarget.com/definition/proxy-server.
[3] Protocolo TCP. Disponível em: http://br.ccm.net/contents/284-o-protocolo-tcp.
[4] Protocolo HTTP. Disponível em: http://br.ccm.net/contents/266-o-protocolo-http.
[5] KUROSE, J. F. e ROSS, K. - Redes de Computadores e a Internet - 5ª Ed., Pearson, 2010.

estruturas.h File Reference

Arquivo principal com a funcao main com servico de proxy. More...

```
#include <netinet/tcp.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <ctype.h>
#include <setjmp.h>
#include <signal.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <errno.h>
#include <math.h>
#include <pthread.h>
#include <semaphore.h>
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in.h>
#include <arpa/inet.h>
```

Go to the source code of this file.

Classes

struct	**headerList**
struct	**requestORresponse**

Macros

#define	**BACKLOG**	20
#define	**BUFFER**	32768

Typedefs

typedef struct **headerList**		**HeaderList**
typedef struct **requestORresponse**		**RequestORResponse**

Functions

void * **connectionHandler** (void *c_pNewSocketFD)

	Atraves de thread maneja pares requesicoes e respostas. More...
HeaderList *	**createHeaderList** () Aloca um ponteiro de estutura HeaderList. More...
int	**emptyHeaderList** (**HeaderList** *list) Verificia se a lista de cabecalho esta vazia. More...
void	**freeHeaderList** (**HeaderList** *list) Libera memoria da estrutura. More...
void	**freeRequestORResponseFiedls** (**RequestORResponse** ***requestORresponse**) Libera a memoria da estrutura que foi passada com parametro. More...
void	**get_1_line** (char *first_l, char *buffer) Pega a primeira linha de uma requisicao. More...
void	**get_status** (char *status, char *buffer) Pega o status da primeira linha de uma requisicao. More...
RequestORResponse *	**getRequestORResponseFields** (char *buffer) Parse da string buffer guarda os campos de HTTP em uma struct. More...
char *	**getRequestORResponseMessage** (**RequestORResponse** ***requestORresponse**) Pega estrutura que tem os campos separados e monta uma string de requisicao. More...
void	**handle_client** (int c_newSocketFD) Maneja a conexao especifica de um cliente. More...
HeaderList *	**insertHeaderList** (**HeaderList** *list, char *headerFieldName, char *value) Insere na lista de cabecalho de acordo com o headerFieldName e a string value. More...
int	**main** (int argc, char *argv[]) Funcao principal do programa. More...
void	**printHeaderList** (**HeaderList** *list) Printa tudo que esta na lista de cabecalho. More...
char *	**search_host** (**RequestORResponse** *c_request) Ṕrocura o valor de host dado a estrutura. More...
int	**verify_status** (char *status) Verifica o status esta OK (2xx). More...

Detailed Description

Arquivo principal com a funcao main com servico de proxy.

Macro Definition Documentation

BACKLOG

```
#define BACKLOG   20
```

BUFFER

```
#define BUFFER   32768
```

Typedef Documentation

HeaderList

typedef struct **headerList HeaderList**

RequestORResponse

typedef struct **requestORresponse RequestORResponse**

Function Documentation

connectionHandler()

void * connectionHandler (void * c_pNewSocketFD)

Atraves de thread maneja pares requesicoes e respostas.

Parameters

c_pNewSocketFD descritor da socket do servidor.

createHeaderList()

HeaderList * createHeaderList ( )

Aloca um ponteiro de estutura HeaderList.

Returns

Ponteiro para HeaderList ja inicializada.

emptyHeaderList()

int emptyHeaderList (**HeaderList** * list)

Verificia se a lista de cabecalho esta vazia.

Parameters

list ponteiro para estrutura que contem os cabecalhos e seus valores.

Returns

1 se estiver vazio, 0 se estiver com elementos.

freeHeaderList()

void freeHeaderList (**HeaderList** * list)

Libera memoria da estrutura.

Parameters

list ponteiro para estrutura que contem os cabecalhos e seus valores.

freeRequestORResponseFiedls()

void freeRequestORResponseFiedls (**RequestORResponse** * requestORresponse)

Libera a memoria da estrutura que foi passada com parametro.

Parameters

requestORresponse estrutura que sera liberada.

get_1_line()

```
void get_1_line ( char *  first_l,
                  char *  buffer
                )
```

Pega a primeira linha de uma requisicao.

Parameters

first_l String que ter apenas o valor da primeira linha.

buffer String que tem todo conteudo do buffer.

get_status()

```
void get_status ( char *  status,
                  char *  buffer
                )
```

Pega o status da primeira linha de uma requisicao.

Parameters

status String que ter apenas o valor do codigo de status.

buffer String que tem todo conteudo do buffer.

getRequestORResponseFields()

```
RequestORResponse * getRequestORResponseFields ( char *  buffer )
```

Parse da string buffer guarda os campos de HTTP em uma struct.

Parameters

buffer -> string que tem os dados enviados pelo browser.

Returns

RequestORResponse estrutura que armazena as strings do cabecalho e formato HTTP.

getRequestORResponseMessage()

char * getRequestORResponseMessage (**RequestORResponse** * requestORresponse)

Pega estrutura que tem os campos separados e monta uma string de requisicao.

Parameters

requestORresponse estrutura que sera usada para montar a requisicao ou mensagem de resposta.

Returns

String no formato correto para envio de requisicao.

handle_client()

void handle_client (int c_newSocketFD)

Maneja a conexao especifica de um cliente.

Parameters

c_newSocketFD descritor da socket do cliente.

insertHeaderList()

HeaderList * insertHeaderList (**HeaderList** * list,
char * headerFieldName,
char * value
)

Insere na lista de cabecalho de acordo com o headerFieldName e a string value.

Parameters

list ponteiro para estrutura que contem os cabecalhos e seus valores.

headerFieldName String com o nome do campo do cabecalho.

value valor do campo.

Returns

Ponteiro para HeaderList que contem o novo elemento inserido.

main()

```
int main ( int      argc,
           char *  argv[]
         )
```

Funcao principal do programa.

Parameters

argc Quantidade de argumentos passados pela linha de comando.

argv[] Argumentos passados pela linha de comando.

Returns

Status de execucao.

printHeaderList()

```
void printHeaderList ( HeaderList *  list )
```

Printa tudo que esta na lista de cabecalho.

Parameters

list ponteiro para estrutura que contem os cabecalhos e seus valores.

search_host()

```
char * search_host ( RequestORResponse *  c_request )
```

Procura o valor de host dado a estrutura.

Parameters

c_request estrutura que sera usada para procurar o valor do host.

Returns

String com o valor do host ou NULL.

verify_status()

int verify_status (char * status)

Verifica o status esta OK (2xx).

Parameters

status String que tem o codigo de status.

Returns

1 se comeca com 2 e 0 caso contrario.

cache.h File Reference

Arquivo com as instrucoes da cache. More...

Go to the source code of this file.

Classes

struct **list_cache**

Macros

#define **MAX_CACHE_SIZE** (10<<20) /* 10MB */

#define **MAX_OBJECT_SIZE** (1<<20) /* 1MB */

Typedefs

typedef struct **list_cache** **cache_object**

Functions

int **add_to_cache** (char *data, int site_size, char *url)
Funcao para adicionar um objeto cache da cache. More...

cache_object * **cache_find** (char *url)
Funcao para procurar um objeto cache de acordo com a url. More...

void **remove_from_cache** ()
Funcao para remover um objeto cache da cache. More...

Variables

int **cache_size**

long int **global_time**

cache_object * **head**

pthread_rwlock_t **lock**

Detailed Description

Arquivo com as instrucoes da cache.

Macro Definition Documentation

MAX_CACHE_SIZE

```
#define MAX_CACHE_SIZE   (10<<20) /* 10MB */
```

MAX_OBJECT_SIZE

```
#define MAX_OBJECT_SIZE   (1<<20) /* 1MB */
```

Typedef Documentation

cache_object

```
typedef struct list_cache cache_object
```

Function Documentation

add_to_cache()

```
int add_to_cache ( char *  data,
                   int     site_size,
                   char *  url
                 )
```

Funcao para adicionar um objeto cache da cache.

Parameters

data	dado da pagina em si.
site_size,tamanho	do site que sera armazenado.
url	string que corresponde a url que sera armazena na cache.

Returns

Retorna 1 se deu certo, zero se deu errado.

cache_find()

cache_object * cache_find (char * url)

Funcao para procurar um objeto cache de acordo com a url.

Parameters

url string que corresponde a url que sera procurada na lista de objetos de cache.

Returns

Objeto cache referente a URL ou NULL se nao foi encontrado.

remove_from_cache()

void remove_from_cache ( )

Funcao para remover um objeto cache da cache.

Variable Documentation

cache_size

int cache_size

Variavel global com o tamanho da cache

global_time

long int global_time

head

cache_object* head

Cabeca da lista de objetos cache

lock

pthread_rwlock_t lock

inspecao.h File Reference

Arquivo com funcoes para o modulo de inspecao. More...

```
#include <stdio.h>
#include <stdlib.h>
```

Go to the source code of this file.

Functions

void **edita_campo_requisicao** ()
Edita um campo de requisicao. More...

void **edita_campo_resposta** ()
Edita um campo de resposta. More...

void **entrega_browser** ()
Manda a resposta para o browser. More...

void **intercepta_requisicao** (**RequestORResponse** *c_request)
Intercepta a requisicao antes da mesma ser enviada pelo proxy. More...

void **intercepta_resposta** (**RequestORResponse** *s_response)
Intercepta a resposta antes da mesma ser ao browser. More...

void **janela_inspecao** ()
Exibe a janela de inspecao. More...

void **janela_requisicao** ()
Exibicao da janela de requisicao. More...

void **janela_resposta** ()
Exibicao da janela de resposta. More...

void **proxy_envia** ()
Manda a requisicao para o proxy. More...

Detailed Description

Arquivo com funcoes para o modulo de inspecao.

Function Documentation

edita_campo_requisicao()

void edita_campo_requisicao ( )

Edita um campo de requisicao.

edita_campo_resposta()

void edita_campo_resposta ( )

Edita um campo de resposta.

entrega_browser()

void entrega_browser ( )

Manda a resposta para o browser.

intercepta_requisicao()

void intercepta_requisicao (**RequestORResponse** * c_request)

Intercepta a requisicao antes da mesma ser enviada pelo proxy.

Parameters

c_request Ponteiro para estrutura que contem os campos das requisicoes HTTP.

intercepta_resposta()

void intercepta_resposta (**RequestORResponse** * c_request)

Intercepta a resposta antes da mesma ser ao browser.

Parameters

c_request Ponteiro para estrutura que contem os campos das requisicoes HTTP.

janela_inspecao()

void janela_inspecao ( )

Exibe a janela de inspecao.

janela_requisicao()

void janela_requisicao ( )

Exibicao da janela de requisicao.

janela_resposta()

void janela_resposta ( )

Exibicao da janela de resposta.

proxy_envia()

void proxy_envia ( )

Manda a requisicao para o proxy.

filtragem.h File Reference

Arquivo com as operacoes de filtragem. More...

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <time.h>
```

Go to the source code of this file.

Macros

#define	**ACCEPT_LOG** 1
#define	**DENY_ERRO** 3
#define	**ERRO_LOG** 2
#define	**MAX_DADO** 6000
#define	**MAX_STR** 500
#define	**NOT_FILTERED** 4
#define	**SIZE_LISTA** 30

Functions

FILE *	**abrindo_arquivo** (char *nome_arquivo) Funcao para abrir um arquivo.txt . More...
FILE *	**abrindo_log** (char *nome_arquivo) Funcao para abrir um arquivo<log>.txt . More...
int	**checkLists** (char *nome_arquivo, char *mensagem) Funcao que checa se uma mensagem esta em um determinado arquivo. More...
int	**denyterms_body** (char *body, char *url) Funcao que checa se em um corpo tem algum termo proibido. More...
int	**denyterms_request** (char *request) Funcao que checa se em uma url tem algum termo proibido. More...
int	**filtragem_url** (char *url) Funcao que checa se uma url se encontra na whitelist, na blacklist, denyterms ou se nao esta nem um destes arquivos. More...
int *	**Length_denyterms** (void) Funcao para guardar a quantidade de elementos e o tamanho de cada elemento do arquivo denyterms.txt. More...
void	**mensagem_log** (char *url, int opcao) Funcao para escrever um log - com data, url e se foi aceita ou rejeitada. More...
void	**mensagem_log_body** (char *url, char *dado) Funcao para escrever um log - com data, url e se foi rejeitada no corpo. More...

Detailed Description

Arquivo com as operacoes de filtragem.

Macro Definition Documentation

ACCEPT_LOG

```
#define ACCEPT_LOG   1
```

DENY_ERRO

```
#define DENY_ERRO   3
```

ERRO_LOG

```
#define ERRO_LOG   2
```

MAX_DADO

```
#define MAX_DADO   6000
```

MAX_STR

```
#define MAX_STR   500
```

NOT_FILTERED

```
#define NOT_FILTERED   4
```

SIZE_LISTA

```
#define SIZE_LISTA   30
```

Function Documentation

abrindo_arquivo()

FILE * abrindo_arquivo (char * nome_arquivo)

Funcao para abrir um arquivo.txt .

Parameters

nome_arquivo corresponde ao nome do arquivo que tera que ser lido.

Returns

Ponteiro para FILE* ja aberto.

abrindo_log()

FILE * abrindo_log (char * nome_arquivo)

Funcao para abrir um arquivo<log>.txt .

Parameters

nome_arquivo corresponde ao nome do arquivo que tera que ser lido.

Returns

Ponteiro para FILE* ja aberto.

checkLists()

```
int checkLists ( char *  nome_arquivo,
                 char *  mensagem
               )
```

Funcao que checa se uma mensagem esta em um determinado arquivo.

Parameters

nome_arquivo corresponde ao nome do arquivo sera lido.

mensagem que sera procurada no arquivo.

Returns

1 se for encontrado, zero se nao for encontrado.

denyterms_body()

```
int denyterms_body ( char *  body,
                     char *  url
                   )
```

Funcao que checa se em um corpo tem algum termo proibido.

Parameters

body corpo que sera verificado.

url que eh passada para funcao de LOG caso tenha termos proibidos.

Returns

1 se for encontrado no denyterms.txt, caso contrario 0.

denyterms_request()

```
int denyterms_request ( char *  request )
```

Funcao que checa se em uma url tem algum termo proibido.

Parameters

request eh a url que sera verificada.

Returns

1 se for encontrado no denyterms.txt, caso contrario 0.

filtragem_url()

```
int filtragem_url ( char *  url )
```

Funcao que checa se uma url se encontra na whitelist, na blacklist, denyterms ou se nao esta nem um destes arquivos.

Parameters

url eh a url que sera filtrada.

Returns

1 se for encontrado no denyterms.txt ou blacklist.txt, caso contrario 0.

Length_denyterms()

int * Length_denyterms (void)

Funcao para guardar a quantidade de elementos e o tamanho de cada elemento do arquivo denyterms.txt.

Returns

Ponteiro para FILE* ja aberto.

mensagem_log()

```
void mensagem_log ( char *  url,
                    int     opcao
                  )
```

Funcao para escrever um log - com data, url e se foi aceita ou rejeitada.

Parameters

url que sera colocada no log.

opcao indica se eh do modo aceito <whitelist ou="" nao="" proibido>=""> ou rejeitado <blacklist, denyterms>.

mensagem_log_body()

```
void mensagem_log_body ( char *  url,
                         char *  dado
                       )
```

Funcao para escrever um log - com data, url e se foi rejeitada no corpo.

Parameters

url que sera colocada no log.

dado string que foi responsavel por ser rejeitada.