Universidade de Brasília
Departamento de ciência da computação
Disciplina: Teleinformática e redes 2 – 1°/2018
Professor: João Gondim

Projeto da disciplina- Um Inspetor HTTP baseado em Proxy Server

Link do repositório GitHUb: https://github.com/klarkgable/Projeto-TR2-1-2018

Alunos:

Klark Gable Souza Porto - 120015421

Gabriel Rodrigues Diógenes Macedo - 15/0126808

INTRODUÇÃO TEÓRICA

1) Cliente-servidor

O modelo cliente-servidor é uma estrutura de aplicação que distribui as tarefas e cargas de trabalho entre os fornecedores de um recurso ou serviço(servidores), e os requerentes dos serviços(clientes).

Geralmente os clientes e servidores comunicam através de uma rede em computadores distintos, mas tanto o cliente quanto o servidor podem residir no mesmo computador.

Um servidor é um host que está executando um ou mais serviços ou programas que compartilham recursos com os clientes. Um cliente não compartilha seus recursos, mas solicita um conteúdo ou função do servidor. Os clientes iniciam sessões de comunicação com os servidores que aguardam requisições de entrada.

Características do Cliente:

Inicia pedidos para servidores;
Espera por respostas;
Recebe respostas;
Conecta-se a um pequeno número de servidores de uma só vez ;
Normalmente interage diretamente com os servidores através de seu software aplicação especifico, que lhe possibilita a comunicação com o servidor;
Utiliza recursos da rede.

Características do Servidor:

Sempre espera por um pedido de um cliente;
Atende os pedidos e, em seguida, responde aos clientes com os dados solicitados;
Podem se conectar com outros servidores para atender uma solicitação específica do cliente; jamais podem se comunicar.
Fornece recursos de rede.
Normalmente interage diretamente com os usuários finais através de qualquer interface com o usuário;

Estrutura o sistema.

2) Requisições HTTP

HTTP é a sigla de "Hyper Text Transfer Protocol", o que traduzido seria "Protocolo de Tranferência de Hiper Texto". Quando acessamos a pígina do google, o navegador se conecta na porta 80 do servidor - isto mesmo, o protocolo HTTP roda sob a porta 80 - e envia uma requisição HTTP para este. Isso não ocorre apenas com páginas WEB - podemos fazer download de arquivos de som, fotos, músicas, etc.

Cliente ----------> Servidor (1)
Cliente <---------- Servidor (2)

(1) O cliente se conecta na porta 80 do servidor; Envia uma requisição HTTP.
(2) O servidor obtém esta requisição, e dependendo do seu conteúdo, envia uma determinada resposta ao cliente.

Esta é a estrutura bísica de uma requisição HTTP. Cada "comando" equivale a um header (ou cabeçalho). Poderíamos resumir da seguinte forma:

[CLIENTE]
cabeçalhos

[SERVIDOR]
cabeçalhos de resposta

(código do arquivo requisitado)

Este header diz ao servidor que o cliente está requisitando o arquivo principal do diretório

3) Sockets

Especificamente em computação, um soquete pode ser usado em ligações de redes de computadores para um fim de um elo bidirecional de comunicação entre dois programas. A interface padronizada de soquetes surgiu originalmente no sistema operacional Unix BSD(Berkeley Software Distribution); portanto, eles são muitas vezes chamados de Berkeley Sockets. É também uma abstração computacional que mapeia diretamente a uma porta de transporte (TCP ou UDP) e mais um endereço de rede. Com esse conceito é possível identificar unicamente um aplicativo ou servidor na rede de comunicação IP.

Um socket identifica univocamente um usuário TCP
Permite a associação entre processos de aplicação
O identificador da porta é concatenado ao endereço IP, onde a entidade TCP está rodando, definindo um socket.
Função socket ()
exemplo de uso
socket (familia, tipo, protocolo);
O nome de um socket sempre está relacionado a um espaço de nomes, também chamado de

domínio (socket domain).Cada espaço de nomes é definido por uma macro na forma PF_* (que vem do termo Protocol Family). Os sockets são divididos em tipos. São eles: SOCK_STREAM, SOCK_DGRAM, SOCK_SEQPACKET, SOCK_RAW, SOCK_RDM, SOCK_PACKET.

4) Servidor Proxy

Em redes de computadores, um servidor proxy é um servidor que atua como intermediário para solicitações de clientes que buscam recursos de outros servidores. Um cliente se conecta ao servidor proxy, solicitando algum serviço, como um arquivo, conexão, página da web ou outro recurso disponível em um servidor diferente, e o servidor proxy avalia a solicitação como uma maneira de simplificar e controlar sua complexidade. Proxies foram inventados para adicionar estrutura e encapsulamento aos sistemas distribuídos.

ARQUITETURA DO PROJETO

O projeto foi feito em C++, usando API padrões e várias bibliotecas de suporte para aplicações em redes. Foi feito em C++, pois acreditamos que na visão orientada a objetos seria melhor estruturado e a comunicação entre classes através dos seus respectivos módulos dariam uma visão mais tangível e simples de entender e interagir.

Para isso, dividimos o projeto em vários módulos:

1) Servidor(módulo para servidorProxy, por onde passam as requisições e respostas)

2) Conexao(módulo geral para sockets e conexao interna e externa)

3) Header(responsável por setar ou recuperar cabeçalhos HTTP de requisição ou resposta)

Obs: os módulos “conexao” e “servidor” em conjunto e utilizando das funções do Header, formam o Inspetor HTTP.

4) ModuloBase(Serve como base para recursos, spider e dump)

5) main(É o módulo principal que chama as rotinas, e seleciona entre inspeção, spider ou cliente recursivo)

6) Para definição das funções e estruturas do projeto, temos o módulo Estrutura.h,

O interceptor tem um lado que recebe e envia para o navegador e um outro lado que recebe e envia para a internet. Nesse caso, o servidor será o responsável por fazer essa comunicação entre os lados.

O spider irá percorrer todo o site procurando por outros html linkados e monta a árvore de referência.

O cliente recursivo irá fazer o dump, que seria parecido com o spider, porém não só com html, e no final baixa para uma pasta local.

Definição das classes:

```
 ///classe para Sockets
classe Socket
Atributos públicos:
 Socket();
 Socket( int Descritor );
 ~Socket();

 //funçoes para atribuiçao padrao para sockets
 bool conecta( std::string nome, std::string porta, int socketFamily = AF_INET, int socketType =
SOCK_STREAM, int protocol = 0 );
 bool percorre( std::string nome, std::string porta, int queueSize = 1, int socketFamily = AF_INET,
int socketType = SOCK_STREAM, int protocol = 0 );

        bool Valido();
        int getDescritor();
        unsigned int getFamily();
        uint16_t getPorta();
        uint32_t getEndereco();
        std::string getStringPorta();
        std::string getStringEndereco();

        bool operator==( Socket &rhs );

Atributos privados:
        bool valido;
        int Descritor;
        struct sockaddr_in endereco;


///definição das estruturas e funções do Header

   //Uma´unica estrutura para cabeçalho http que identifica e separa os campos para tratar de
requisiçao e resposta
   struct Header {

      //metodo para separar os campos
      Header( std::string& palavra );

      //metodo que transforma struct para string para envio na rede
         std::string texto( bool incluiCorpo = true );

      //primeira linha e campos http
         std::string PrimeiraLinha;
         std::vector< field > campos;

      //corpo da mensagem http
         std::string corpo;

      // caminho e porta
         std::string host;
```

```
        std::string porta

///definiçoes das estruturas e funções de conexao interna
classe conexaoInterna

   Atributos publicos:
      conexaoInterna( int porta );
      ~conexaoInterna();
         typedef std::tuple< std::weak_ptr< Socket >, HTTP::Header > Requisicao;


         bool aceitaConexoes();
         bool recebeRequisicoes();
         void fechaSocket( int Descritor );
ssize_t enviaResposta( std::weak_ptr< Socket > recebendoSocket, HTTP::Header resposta );

         std::vector< Requisicao > requisicoesRecebidas;

   Atributos privados:
         Socket escutandoSocket;
         std::vector< std::shared_ptr< Socket > > socketsConectados;


///definiçoes das estruturas e funções de conexao externa
classe conexaoExterna
   Atributos publicos:
      conexaoExterna(int porta);
         conexaoExterna();
      ~conexaoExterna();
         typedef std::tuple< std::weak_ptr< Socket >, HTTP::Header > Resposta;

         ssize_t enviaRequisicao( std::weak_ptr< Socket > requisitandoSocket, HTTP::Header
requisicao );
         bool recebeRespostas();

         std::vector< Resposta > respostasRecebidas;

   Atributos privados:
      typedef std::tuple< std::shared_ptr< Socket >,std::weak_ptr< Socket > > parSocket;
         int acharParSocket( std::weak_ptr< Socket > s_w_ptr );
         int acharParSocket( std::shared_ptr< Socket > s_s_ptr );
         void arrumarSockets();

         std::vector< parSocket > socketsCriados;


 ///definição das estruturas e funções do Servidor Proxy

classe ServidorProxy

   Atributos publicos:
      ServidorProxy( int porta );
```

```
        ~ServidorProxy();
        bool Loop();

            bool continuando;
            std::vector< conexaoInterna::Requisicao > &requisicoesRecebidas;
            std::vector< conexaoInterna::Requisicao > requisicoesEnvio;
            std::vector< conexaoExterna::Resposta > &respostasRecebidas;
            std::vector< conexaoExterna::Resposta > respostasEnvio;

      Atributos privados:
            conexaoInterna ci;
            conexaoExterna ce;


///definição das estruturas e funções do Spider e Cliente recursivo
struct Recurso {

Atributos publicos:
            typedef std::tuple< unsigned long long int, unsigned long long int, std::string > Referencia;
                        Recurso( std::string host, std::string nome, std::string respostaHTTP );

                        std::string getNome();
                        std::string getNomeLocal();
                        std::vector< Referencia > getRecursosReferenciados();
                        void setaReferencias( std::vector< long long int > refs );
                        bool salvar( std::string caminhoRoot );
                        bool Valido();


            Atributos privados:
                        void procuraReferencias( const char* propriedadeHTML );
                        std::string host;
                        std::string nome;
                        std::string nomeLocal;
                        std::string dados;
                        std::vector< Referencia > recursosReferenciados;
                        std::vector< unsigned long int > referencias;
                        bool valido;


classe  Spider
            Atributos publicos:
                        Spider( std::string host );
                        bool Valido();
            Atributos privados:
                        std::string recursosBaixados( std::string host, std::string nomeRecurso );
                        long long int achaRecursos( std::string nomeRecurso );

                        Socket socket;
                        bool sucesso;
                        std::string nomeArvoreRoot;
                        std::vector< Recurso > arvore;
```

COMO RODAR O PROJETO

Para rodar projeto, é bem simples. Sistema usado tem que ser distribuição Linux.
Ter instalado o compilador G++ e o Make para rodar o arquivo makefile

Passos:

1)Baixar arquivos fontes do projeto, copiar para alguma pasta;

2)Abrir terminal e caminhar até a pasta do projeto;

3)Rodar arquivo makefile(comando "make -f makefile");

4)Será gerado os arquivos objetos e o executável “aracne”;

5)Rodar aracne(comando ./aracne) que irá rodar na porta 8228 caso não passe outra porta como argumento. Para fazer os requests e responses o navegador web que for utilizar deve ter o proxy configurado no servidorProxy:127.0.0.1 e porta 8228.

6) Escolha opção 1 para rodar o interceptor ou opção 2 para rodar o spider. Dumper não foi possível implementar

7) No caso do interceptor, abra o navegador e entre nos sites para gerar as requisições e respostas.

6)Para rodar o Spider, precisa abrir a main e setar a chamada ao Spider com por exemplo Spider("www.sitequalquer.com"). Caso não faça isso ele irá rodar o spider no site configurado previamente.