Servidor de Proxy
Implementação com Cache

Lúdio A. Oliveira[1], **Weber Mourão** [1]

[1]Faculdade de Computação - Universidade Federal de Mato Grosso do Sul (UFMS)
Campo Grande - MS - Brazil

2

ludio.ao, weber.mourao@gmail.com

Resumo. *Este relatório descreve o trabalho realizado para a disciplina de Redes de Computadores, onde a nossa tarefa principal seria desenvolver um proxy que retorna requisições de um servidor web HTTP anteriormente implementado, em modo de multi-processo e multi-threading.*

1. O Proxy

Um servidor proxy é um computador que atua como intermediário entre uma rede local e a Internet. Por exemplo, uma empresa que tem um link Internet em apenas um computador pode instalar um servidor proxy neste computador, e todos os outros podem acessar a Internet através do proxy. Além disso, ele pode fazer ”cache”, armazenando localmente as páginas mais consultadas, o que torna o acesso a elas mais rápido. Hoje em dia, muitos servidores proxies também fazem a função de *firewall*, acrescentando segurança ao acesso à Internet da empresa (pois todos passam a acessar a Internet através de uma porta única.

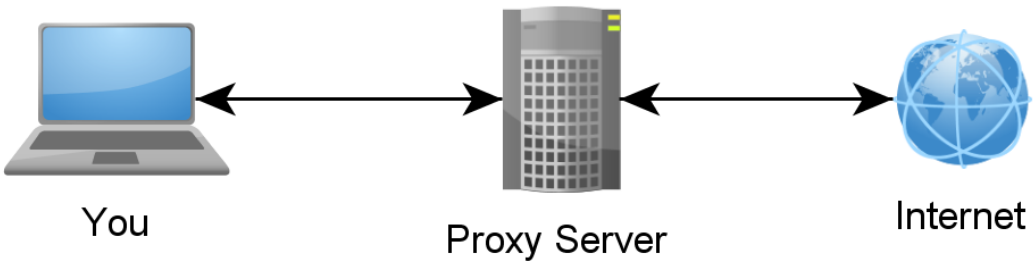

Figura 1. Funcionamento do Servidor Proxy.

2. Implementação do Servidor Proxy

Nesta versão de implementação, o servidor aceita apenas requisições HTTP/1.0 e HTTP/1.1, de acordo com a RFC 2616, e só responde por métodos *GET*. Além disso, não aceita requisições *HTTPS*.

2.1. Modo Multi-threading

Em modo de *multi-threading* a função *HTTPServer::RunMultiThreaded()* inicia a thread principal, e opera em modo de escuta (*listening*) até que se receba uma nova conexão.

Todas as *threads* são criadas sob demanda. Enquanto não recebe nenhuma requisição, a *thread* fica em *stand-by* até receber novas requisições.

A cada requisição recebida, uma nova *thread* é criada, e então a conexão é tratada pela classe *HttpRequest* onde acontece o *parser* da requisição (cabeçalho) e devolve um arquivo lido.

No método *HandleGet*, ele trata as requisições GET e repassa para o método *downloadFile* que por sua vez verifica se a URL está na *cache*, se estiver, então ele retorna o arquivo lido, caso contrário, ele cria uma conexão com o servidor alvo, e então, faz a requisição do caminho solicitado pelo cliente. Após receber o conteúdo completo, então ele fecha a conexão com o servidor e também encerra a threads.

Quando o servidor é encerrado, todas as *threads* associadas e que ainda estão sendo tratadas pela *thread* principal são finalizadas.

3. Inicialização do servidor

Como dito anteriormente, o servidor proxy opera no modo de escuta utilizando-se de threads.

Para compilar a aplicação, rode o arquivo *MakeFile* no terminal. A aplicação gerada é denominada de proxyHttp.

Para iniciar em modo de operação do tipo multi-threading, faça:

```
~/.bash > proxyHttp [PORTA] [MB]
```

PORTA: número da porta em que o socket / proxy irá inicializar. MB: tamanho da cache em megabytes.

Todas a funcionalidades descritas no trabalho foram implementadas.

4. Problemas encontrados

No trabalho do HTTP era enviado uma *string*, não tratávamos o caso de arquivos binários. O que foi um motivo e tanto para quebrarmos cabeça ao longo dos dias, e depois notar que realmente era o *buffer* que deveria ser enviado.

Outro problema é que em algum momento que removemos um arquivo da *cache*, ele pode estar sendo utilizado por outro, não encontramos a solução para tentar resolver isso. Pensamos em usar um *mutex* para fazer o bloqueio, mas isso causaria problema na performance.

5. Conclusão

Este trabalho ajudou bastante no entendimento de requisição para um servidor via socket, e como os dados são tratados pelos servidores HTTP, ex: Apache, IIS, etc. Também aperfeiçoamos o nosso entendimento acerca do assunto de *sockets*.