	UNIVERSIDADE FEDERAL DA PARAIBA - CAMPUS I	
	DEPARTAMENTO	SISTEMAS E COMPUTAÇÃO
	DISCIPLINA	Top. Esp. em C.C. – Desenvolvimento Web
	PERÍODO:	X 14:00/18:00
	PROFESSOR	Raoni Kulesza SEMESTRE: 2018.1

PRÁTICA 1

1 ATIVIDADE

Implementar um Servidor Web a partir de dois arquivos base. Na classe **Servidor** deve estar implementado o Servidor e na classe **Conexao** o socket que irá comunicar-se com o cliente segundo o protocolo HTTP. Para verificar o funcionamento do servidor, você poderá escolher como cliente do seu servidor um Browser (navegador) da sua preferência.

Baseado na seção TEORIA deste documento, a solução desenvolvida deverá:

- 1) suportar o MÁXIMO de códigos de respostas do protocolo HTTP (ver referencia 1 na seção final deste documento);
- 2) comportar *Virtual Hosting*;
- 3) implementar um mecanismo de autenticação de usuários para proteger pastas específicas;
- 4) desenvolver um mecanismo de log de acesso;
- 5) aceitar múltiplas conexões de clientes.

IMPORTANTE

- Para cálculo da nota da seleção cada item acima (de 1 a 5) terá **peso 2 (dois)** com uma nota que irá de 0 (zero) a 10 (dez).
- Esta atividade é INDIVIDUAL. O código do projeto tem que ser 100% implementado pelo aluno. NÃO podem ser empregadas bibliotecas e APIs de terceiros, apenas pacotes oficiais do J2SE (*Java Standard Edition*),

2 TEORIA

2.1 Camada de Aplicação em Redes TCP / IP.

Seguindo o modelo OSI-ISO, podemos dizer que uma camada em uma arquitetura de rede , sempre presta serviços para uma camada superior. A camada de aplicação é a única camada que não presta serviço para outras camadas, por ser a última. Assim, sua função principal é disponibilizar aplicações aos usuários ou à outras aplicações utilizando as camadas inferiores de rede para prestar este serviço.

A Figura 1 apresenta uma representação gráfica das camadas existentes na arquitetura TCP / IP.

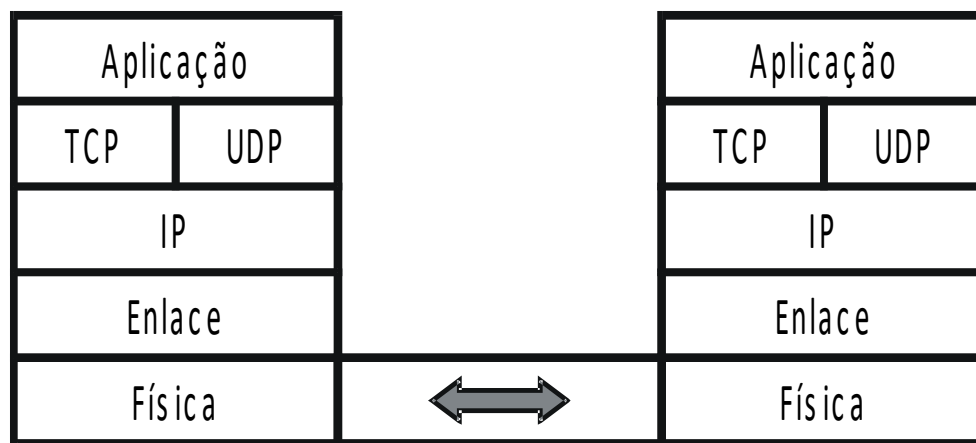


Figura 1 - Camadas do Modelo TCP / IP

A camada de aplicação define os seguintes serviços básicos:

- Estabelecimento, manutenção e liberação de associação entre aplicações;
- Transferência de dados;
- Processamento remoto;
- Sincronização de aplicações;
- Controle de processos.

Como exemplo desses serviços temos:

- Emulação de terminal;
- Correio Eletrônico;
- Transferência, acesso e gerenciamento de arquivos;
- Banco de Dados;
- Gerenciamento de Redes.

2.2 Cliente – Servidor

✍ A arquitetura cliente-servidor é um modelo para interação entre processos concorrentes em execução. O Cliente e o Servidor são entidades lógicas separadas que operam em conjunto, através da troca de mensagens, para realizar um trabalho, podendo coexistir em uma mesma máquina ou em máquinas diferentes.

De uma forma simplificada, podemos dizer que: o cliente é um processo que requisita um serviço a um servidor através de mensagens, e o servidor é um processo que disponibiliza serviços para um cliente através de mensagens.

A Tabela 1 mostra as principais diferenças entre aplicações clientes e servidoras.

Atributo	Cliente	Servidor
Modo de Operação	Ativo – sempre gera requisição de serviços	Reativo – sempre responde a requisição de um serviço
Execução	Possui início e fim	Fica em execução permanente
Objetivo principal	Atender as necessidades do usuário	Prover serviços aos clientes
Transparência	Oculta rede e servidores	Oculta detalhes da implementação de serviços
Inclui	Comunicação com diferentes servidores	Comunicação com diferentes clientes
Exclui	Comunicação entre clientes	Comunicação entre servidores

Tabela 1 - Características de Clientes e Servidores

2.3 Sockets

Sockets são API's (*Application Program Interface*) para acesso aos serviços da camada de transporte numa arquitetura TCP / IP.

Um socket pode solicitar serviços da camada de transporte utilizando os dois protocolos disponíveis nesta camada: UDP (não orientado a conexão) e TCP (orientado a conexão).

Estas interfaces utilizam conceitos comuns às operações com arquivos, de modo que, enviar e receber dados pareçam operações de leitura e escrita em arquivos.

Um socket utilizado para uma aplicação servidora conta com algumas funções a mais que um socket utilizado em um cliente. Esta diferença fica clara na Figura 2, que ilustra o funcionamento de um socket cliente e um socket servidor para uma conexão TCP.

A tecnologia Java dá suporte a sockets por meio das classes do pacote `java.net`.

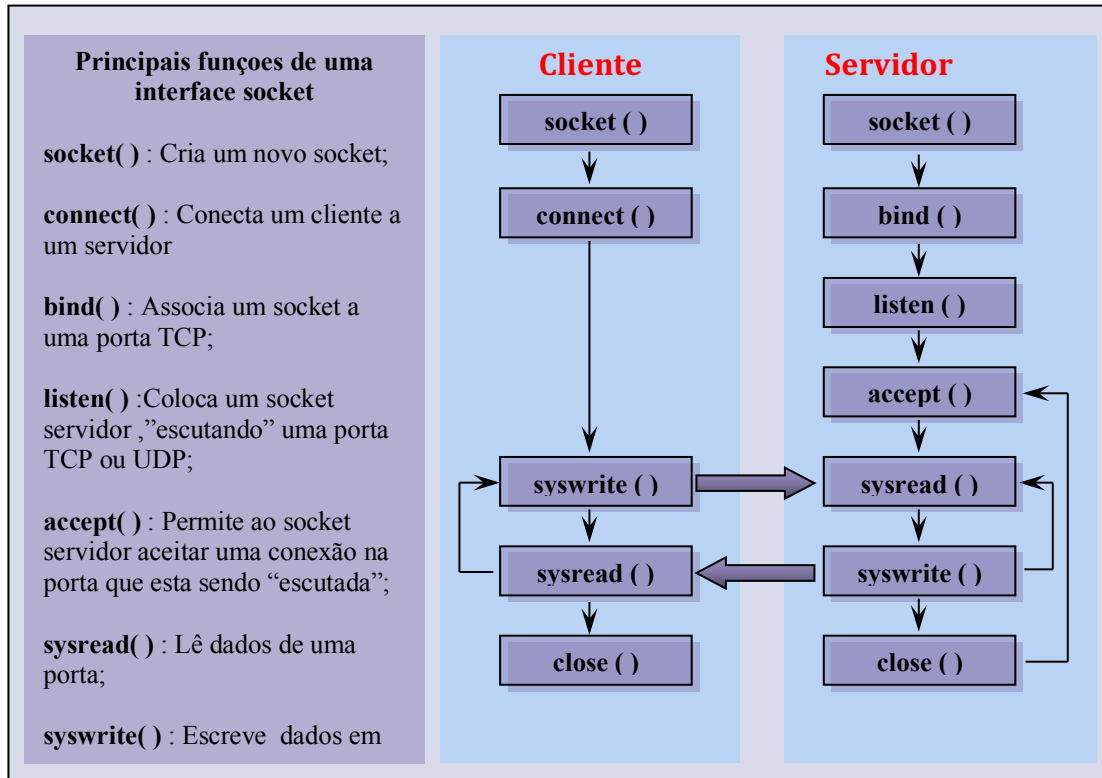


Figura 2 – Funções de sockets servidor e cliente TCP

2.4 HTTP – Hypertext Transfer Protocol

Hypertext Transfer Protocol é um protocolo definido para comunicação entre clientes e servidores Web. Não tem memória ou estados e especifica como o cliente estabelece uma conexão com o servidor, como o cliente pede dados ao servidor, como o servidor responde ao pedido e como é finalizada uma conexão.

Este protocolo utiliza uma conexão TCP para transferir dados e adota o padrão MIME para codificar estes dados.

2.4.1 Transação Típica do protocolo HTTP.

Quando um usuário especifica uma URL no Browser, este inicia uma série de operações que resultam na obtenção do documento especificado.

Uma URL tem o seguinte formato:

protocolo://servidor.dominio:porta/caminho

O protocolo no caso da web é HTTP, servidor é o servidor que hospeda o documento, domínio é o domínio onde se encontra o servidor, porta é onde será feita a

Mime: Multipurpose Internet Mail Extensions

É um padrão para codificação de diversos tipos de dados - como imagens, sons, textos - a serem transmitidos por uma conexão ASCII de 7 bits. Este padrão permite que o tipo de dado enviado seja reconhecido e portanto apresentado corretamente no seu destino. Como o nome diz, ele foi criado para facilitar a transferência de

conexão (se não for especificada, o default é 80) e caminho é um caminho que determina o documento no diretório.

Neste protocolo existem 4 passos para que um cliente possa receber dados de um servidor:

2.4.2 Estabelecer uma conexão TCP com o servidor

Antes de solicitar ou enviar qualquer informação, o cliente deve primeiramente estabelecer uma conexão TCP com o servidor na porta em que ele está “escutando” (esperando pedidos).

Normalmente, servidores Web recebem requisições na porta 80, porém nada impede de um servidor Web “escutar” (esperar requisições) numa outra porta livre. A única restrição é que a porta deve ser maior que 1024.


2.4.3 Requisitar informações

Após estabelecida a conexão com o servidor, o cliente faz uma requisição de dados ao servidor. Um exemplo típico de requisição é este:

```
GET /index.html HTTP/1.0
User-Agent: Mozilla/3.0Gold (WinNT; I)
Host: hypothetical.ora.com
Accept: image/gif, image/x-bitmap, image/jpeg, */*
```

Veja em:

```
http://www.delorie.com/web/index.html
http://www.delorie.com:81/some/url.html
```

 A primeira linha sempre especifica o método (um comando no protocolo HTTP) que deve ser usado e em qual arquivo este método deve ser aplicado. Além disto, informa ao servidor a versão do protocolo utilizado.

Esta é a linha mais importante da mensagem, sendo muitas vezes necessária apenas ela, para que o cliente receba a informação requisitada. A tabela 2 mostra os principais métodos disponíveis no protocolo.

Método	Função	Exemplo
GET	Pede um documento	GET /index.html HTTP/1.0
HEAD	Pede informações sobre um documento	HEAD /index.html HTTP/1.0
POST	Indica que o cliente esta fornecendo informações. Geralmente parâmetros para um arquivo executável.	POST /cgi-bin/creat.pl HTTP/1.0
PUT	Informa que o servidor deverá armazenar as informações que o cliente esta enviando.	PUT /index.html HTTP/1.0

Tabela 2 - Métodos básicos do protocolo HTTP

Analisando a mensagem linha por linha, temos:

GET /index.html HTTP/1.0

O servidor deve usar o método GET para enviar o arquivo index.html do servidor conectado e o browser utiliza o protocolo HTTP 1.0

Da segunda linha da mensagem até a primeira linha em branco vem o cabeçalho da mensagem. No cabeçalho são passadas informações sobre o cliente para o servidor. Geralmente o cabeçalho é opcional, mas muitas vezes estas informações são necessárias para que o cliente receba uma resposta positiva para seu pedido, por exemplo, num caso de autenticação de cliente.

User-Agent: Mozilla/3.0Gold (WinNT; I)

O cliente informa que é um browser Mozilla 3.0 Gold

Host: hypothetical.ora.com

O cliente informa o servidor contactado

Accept: image/gif, image/x-bitmap, image/jpeg, */*

O cliente informa os tipos de codificação MIME aceitas pelo Browser.

Em alguns servidores WWW, quando o nome do arquivo não é especificado na URL, eles passam para o cliente o arquivo index.htm.

Para terminar uma mensagem o cliente acrescenta uma linha em branco após o header.

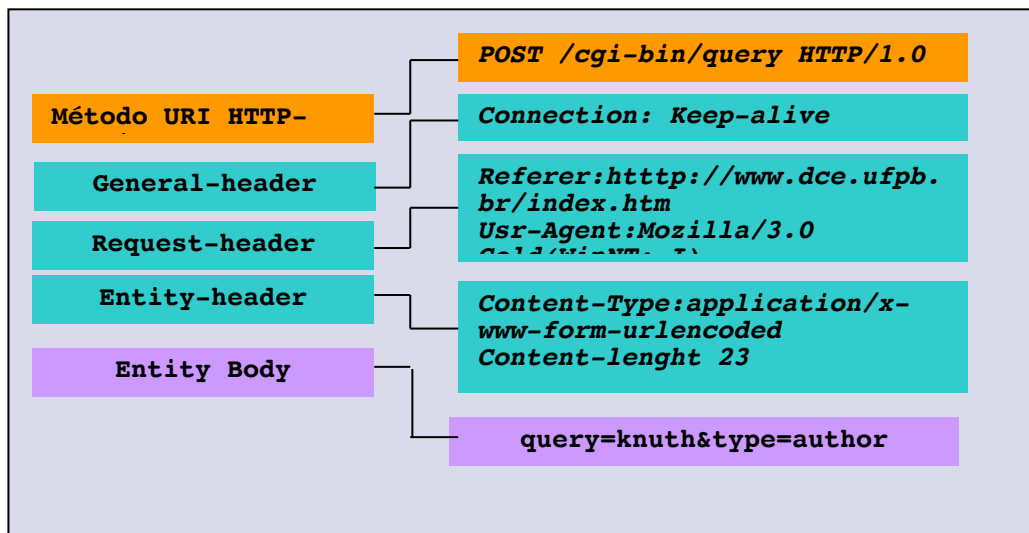


Figura 3 - Formato típico da mensagem de um cliente para um servidor em HTTP

Após a linha em branco, que segue o cabeçalho, o cliente coloca os dados que deseja enviar ao servidor. Neste exemplo isto não acontece pois estamos utilizando o método GET que apenas solicita dados.

2.5 Resposta do servidor

Quando o servidor recebe um pedido de um cliente, responde enviando a informação requisitada ou negando o pedido. Para o pedido feito no item anterior, uma possível resposta seria:

```
HTTP/1.0 200 OK
Server: NSCA/1.4.2
MIME-version: 1.0
Content-type: text/html
Content-length: 46
```

```
<html>
<Title>
A pagina HTML simples
</Title>
</html>
```

Veja em:

```
http://www.delorie.com/web/index.html
http://www.delorie.com/web/headers.html
```

Analisando a mensagem linha por linha temos:

HTTP/1.0 200 OK

Na primeira linha temos a versão do protocolo utilizado pelo servidor e o código de resposta (neste caso 200 significa OK). O servidor sempre utiliza a versão HTTP mais próxima da versão HTTP que o cliente está utilizando.

Server: NSCA/1.4.2

O servidor informa ao cliente o nome e a versão do software utilizado.

MIME-version: 1.0

O servidor informa a versão do padrão de codificação utilizado.

Content-type: text/html

O servidor informa o tipo de conteúdo enviado no corpo da mensagem.

Content-length: 46

O servidor informa a quantidade de caracteres enviados no corpo da mensagem.

A tabela 3 especifica as faixas de resposta que o servidor pode dar e o seu significado geral. A tabela 4 apresenta em detalhes os códigos mais comumente recebidos como respostas a requisições de clientes.

Código de Resposta	Significado
100 – 199	Informativo
200 – 299	O pedido foi bem sucedido
300 – 399	Pedido redirecionado. Necessidade de ação posterior
400 – 499	Pedido incompleto
500 – 599	Erro do servidor

Tabela 3 - Códigos de resposta do protocolo HTTP

Código de Resposta	Significado	Significado
200	OK	A requisição do cliente foi bem sucedida
400	Bad Request	Indica erro de sintaxe na requisição
401	Unauthorized	Requisição não autorizada. Exige autenticação de cliente.
404	Not Found	Não existe o documento especificado pela URL
405	Method Not Allowed	O método especificado não é permitido para a URL especificada
406	Not Acceptable	A URL existe mas não no formato especificado
500	Internal Server Error	Indica uma falha de funcionamento do servidor
501	Not Implemented	Indica que a ação solicitada não foi implementada no servidor

Tabela 4 - Códigos de resposta do protocolo HTTP

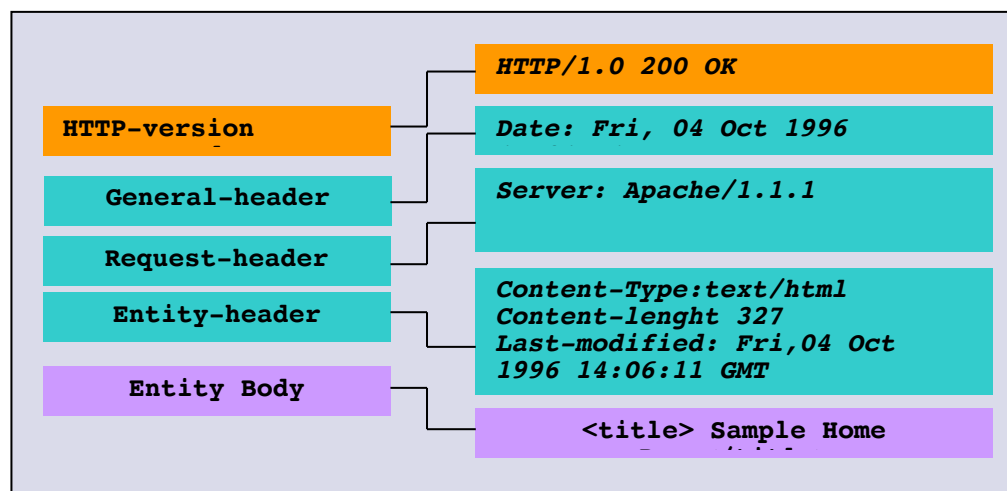


Figura 4 - Formato típico de uma mensagem de um Servidor para um cliente em HTTP

2.6 Fechar a conexão

Na versão 1.0 do protocolo HTTP, após cada resposta a conexão é fechada. Neste caso tanto o cliente como o servidor podem fechar a conexão.

É importante lembrar que como o protocolo, na versão 1.0, não tem estados, se um cliente se conectar novamente a um servidor, o servidor não sabe nada sobre uma conexão anterior deste cliente. Um protocolo que não retém informações sobre uma requisição anterior é chamado de *stateless* – sem memória.

Um exemplo de protocolo com memória é o FTP que pode processar vários pedidos em uma só conexão.

A versão 1.1 do protocolo HTTP prevê memória, de modo que as conexões não serão fechadas após cada pedido. Isto diminui o overhead da rede em função da necessidade de se estabelecer novas conexões.

2.7 Identificação de Clientes e Servidores com HTTP

Cliente e servidores gostam de saber com quem estão trocando informações. Muitas vezes é necessário identificar um cliente ou um servidor para que uma operação se realize.

O protocolo HTTP fornece recursos para identificação de clientes e servidores através dos cabeçalhos de pedidos e respostas.

Um cliente pode se identificar através do cabeçalho *User-Agent* no pedido ao servidor.

```
GET /index.html HTTP/1.0
User-Agent: Mozilla/3.0Gold (WinNT; I)
Host: hypothetical.ora.com
Accept: image/gif, image/x-bitmap, image/jpeg, */*
```

A linha sombreada acima identifica o cliente como: Mozilla 3.0 Gold

Um servidor pode se identificar através do cabeçalho *Server* na resposta ao cliente.

```
HTTP/1.0 200 OK
Server: NSCA/1.4.2
MIME-version: 1.0
Content-type: text/html
Content-length: 46
```

A linha sombreada acima identifica o servidor como: NSCA/1.4.2

2.8 Autenticação de usuários

Muitos dos documentos disponíveis na Web tem seu acesso restrito. Neste caso é necessário a identificação dos usuários para que o documento seja liberado. Nesta situação é utilizado o cabeçalho *Authorization*.

Este cabeçalho tem o seguinte formato: **Authorization:** SCHEME REALM onde Scheme (Esquema de codificação) em HTTP é normalmente do tipo BASIC, que tem uma credencial no formato username:password codificada em base64. Por exemplo, para um usuário webmaster com uma senha “zrqma4v” o cabeçalho *Authorization* é:

```
Authorization: BASIC d2VibWFzZdGVyOnpcW1Hnhy=
```

O parâmetro **Realm** é do tipo de usuário que pede autorização: No exemplo acima o usuário é **System Administrator**.

Um processo típico de autenticação de usuário é o seguinte:

1. O browser requisita um documento ao servidor sem o cabeçalho Authorization;

```
GET /index.html HTTP/1.0
User-Agent: Mozilla/3.0Gold (WinNT; I)
Host: hypothetical.ora.com
Accept: image/gif
Accept: image/x-bitmap
Accept: image/jpeg
Accept: */*
```

2. O servidor coloca na sua resposta o cabeçalho WWW-Authenticate para que o cliente identifique o documento como um documento de acesso restrito;

```
HTTP/1.0 401 Unauthorized
Server: NCSA/1.4.2
MIME-version: 1.0
Content-type: text/html
WWW-Authenticate: Basic realm= "System Administrator"
```

Neste momento o browser mostra na tela uma janela para que o usuário coloque *login* e senha.

3. O browser faz uma nova requisição com o cabeçalho Authorization;

```
GET /index.html HTTP/1.0
User-Agent: Mozilla/3.0Gold (WinNT; I)
Host: hypothetical.ora.com
Accept: image/gif
Accept: image/x-bitmap
Accept: image/jpeg
Accept: */*
Authorization: Basic viW345deErf
```

4. O Servidor envia uma resposta positiva se a senha enviada estiver correta ou negativa se a senha estiver errada.

```
HTTP/1.0 200 OK
Server: NCSA/1.4.2
MIME-version: 1.0
Content-type: text/html
```

```
<html> ...</html>
```

2.9 Arquivos de Log

Servidores WWW normalmente mantêm um arquivo de log, onde estão armazenadas informações sobre os clientes que acessaram seus serviços.

A maioria dos servidores Web adotam um padrão comum para o arquivo de log. Uma linha típica deste padrão tem esse formato:

```
205.160.186.76 unknown - [01/Jan/1996:22:53:58 -0500]  
"Get /bgs/greenbg.gif HTTP/1.0" 200 50
```

Isto quer dizer que o browser do IP 205.160.186.76 requisitou o arquivo /bgs/greenbg.gif deste servidor às 22:53 horas e 58 segundos no dia primeiro de janeiro de 1996. O arquivo foi encontrado e 50 bytes foram transmitidos corretamente para o cliente.

2.10 Virtual Host ou Multihoming

Alguns hosts podem responder a mais de um nome de máquina (*hostname*). Isto quer dizer que um host pode ter mais de um nome associado a seu IP e pode responder diferentemente de acordo com o nome solicitado.

Imagine um servidor respondendo a dois nomes diferentes host0.com e host1.com e que tenha os seguintes diretórios disponíveis para os clientes acessarem: host0 e host1.

Nesta situação, o cliente deve acrescentar na requisição o cabeçalho Host para identificar o diretório desejado. Exemplo:

```
GET /index.html HTTP/1.0  
User-Agent: Mozilla/3.0Gold (WinNT; I)  
Host: host0.com  
Accept: image/gif  
Accept: image/x-bitmap  
Accept: image/jpeg  
Accept: */*
```

A linha sombreada informa à aplicação que o servidor deve pegar o arquivo index.html a partir do diretório raiz host0. Se tivéssemos uma requisição com o cabeçalho:

```
Host: host1.com
```

O arquivo index.html seria fornecido a partir do diretório raiz host1.

2.11 Servidor que Aceita Múltiplas Conexões

Grande parte dos servidores Internet que existem atualmente, possuem a característica de aceitar múltiplas conexões. Isto significa que o servidor pode tratar mais de uma conexão de uma vez.

Para que isto seja possível, o sistema operacional onde o servidor reside deve ter suporte a “*Multithreading*”, que é a capacidade de um simples processo (ou programa) gerar múltiplas,

simultâneas linhas de execução (thread). “*Multithreading*” é similar ao multiprocessamento, entretanto, no “*multithreading*” existe compartilhamento de mesma área de memória.

Todas a threads possuem características comuns, como a seqüência de instruções que devem ser executadas, possuem um estado que pode ser: executando, inicializada, terminada, suspensão, entre outros. As threads também podem possuir níveis de prioridade, que determinam o comportamento do processador com relação a elas.

Um servidor “*multithreading*” pode gerar uma nova thread para cada pedido de um cliente, e logo a seguir já estar preparado para uma nova conexão, possibilitando desta maneira, uma aumento da performance do servidor.

3 REFERENCIAS

1. Códigos de respostas de um servidor HTTP

<http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>

2. Java Network Programming

Elliotte Rusty Harold

Editora: O'Reilly

3. Web Client Programming with Perl

Clinton Wong

Editora: O'Reilly