# EA872: Laboratório de Programação de Software Básico
# Atividade 4

Eleri Cardozo
Ivan L. M. Ricarte
Maurício Ferreira Magalhães
Marco Aurélio Amaral Henriques

Departamento de Engenharia de Computação e Automação
Faculdade de Engenharia Elétrica e de Computação
Universidade Estadual de Campinas

## 1 Objetivos das atividades

As atividades que começarão a ser realizadas a partir de hoje resultarão no desenvolvimento de um elemento da arquitetura WWW (World-Wide Web): o servidor HTTP (Hypertext Transfer Protocol). Esse servidor, apesar de simples quando comparado aos servidores HTTP comerciais, será capaz de:

- interagir com navegadores-padrão, como Firefox e Chrome, por exemplo;
- manter um diretório de páginas HTML (Hypertext Markup Language);
- responder a requisições HTTP.

Tais atividades cobrirão os seguintes temas ministrados na disciplina EA876/EA877/EA879:

- linguagem C: o servidor será desenvolvido em C;
- compiladores: será construído um *parser* para análise das mensagens HTTP submetidas ao servidor utilizando as ferramentas `lex` e `yacc`;
- sistemas operacionais: várias chamadas de sistema serão empregadas para:
  - controlar o diretório de páginas HTML (sistema de arquivos);
  - criar e controlar processos para atender concorrentemente a requisições HTTP submetidas ao servidor (controle de processos);
  - utilizar um conjunto de áreas de memoria na transferência de arquivos para o cliente (gerenciamento de memória).

Adicionalmente, o aluno irá utilizar as chamadas de sistema para acesso à rede (biblioteca de `sockets`), familiarizando-se assim com os conceitos básicos de programação distribuída.

## 2 Visão Geral da World-Wide Web

A rede de informação mundial World-Wide Web (WWW) foi a primeira concretização de propostas de integração de informação espalhada por todo o mundo. A WWW é descrita como uma "iniciativa de recuperação de informação hipermídia em área ampla objetivando dar acesso universal a um grande número de documentos." Efetivamente, a WWW implementou — através de mecanismos uniformes de descrição e acesso — um espaço abstrato de conhecimento no topo da Internet, de forma a integrar milhões de usuários espalhados por muitos países.

A WWW foi inicialmente proposta em março de 1989, dentro do contexto de um centro de pesquisa em física de alta energia (CERN), que agrega membros de diversos países europeus. O objetivo inicial da proposta era:

- prover uma interface consistente de acesso à informação independente da plataforma;
- permitir o armazenamento e recuperação de informação através recursos hipertexto/hipermídia;
- permitir o acesso universal (a partir de qualquer ponto) à informação (a Internet é utilizada como mecanismo de transporte de informação).

Essa iniciativa espalhou-se com taxas de crescimento superiores a qualquer outro serviço suportado pela Internet. Mosaic foi a primeira interface gráfica de interação com a WWW, tendo sido desenvolvida no Centro Nacional para Aplicações em Supercomputação (NCSA) dos Estados Unidos. A primeira versão deste programa tornou-se publicamente disponível no início de 1993. Desde então, diversas interfaces gráficas (navegadores, visualizadores ou *browsers*) para a apresentação de documentos da WWW estão disponíveis, suportando interações com o usuário (dirigidas por mouse), apresentação de documentos hipertexto e hipermídia, conexão com aplicativos externos para a apresentação de imagens, vídeo e áudio em diversos formatos, interação através de mecanismos de formulários (preencher campos, checar opções) e conexão com outros serviços Internet. Também em 1993 foi estabelecido o Consórcio WWW (W3C) com a participação inicial do CERN, MIT[1], Keio University e INRIA[2]. Hoje o Consórcio W3C conta com centenas de membros, estando dentre eles todas as principais empresas do ramo de informática.

A WWW funciona segundo o modelo cliente-servidor. Um servidor Web é um programa sendo executado em um computador cujo propósito principal é servir documentos a outros computadores que façam pedidos. Um cliente Web é um programa que suporta a interface com o usuário e requisita documentos a servidores. O suporte nos servidores e clientes para funções de criptografia e autenticação de usuários permite a utilização da arquitetura WWW em aplicações comerciais tais como *home banking* e comércio eletrônico.

A WWW ampliou sobremaneira a utilização da Internet por tornar a informação mais rica via recursos hipermídia/multimídia além de tornar o acesso à informação mais natural e eficaz. Exemplos de atividades que a WWW viabilizou:

- negócios em geral: compras, catálogos, movimentação bancária;
- informação: jornais e revistas eletrônicas, *news updating*;
- entretenimento: jogos, *chat*;
- cultura e educação: museus e bibliotecas *on-line*, documentação histórica.

[1] Massachusets Institute of Technology.
[2] Institut National de Recherche en Informatique et en Automatique.

Várias empresas, utilizando e aprimorando a tecnologia original do CERN e NCSA, estabelecem novos horizontes para a WWW e, por conseguinte, para a Internet nos seguintes ramos:

- software de comunicação, navegadores;
- acessórios (*plug-ins*) para navegadores;
- provedores de informação;
- provedores de acesso;
- projeto e implementação de sítios WEB.

No caso de navegadores, a Microsoft Corp. dominou inicialmente o mercado com o Internet Explorer/Edge (88% em 2003; 4% em 2023). Hoje o browser dominante é o Chrome da Google (66%). Um dos mais antigos navegadores, o Netscape da (já extinta) Netscape Communications Corp., se tornou a base para várias alternativas de browsers. A Fundação Mozilla vinha aumentando sua participação nesse mercado com o navegador gratuito Firefox. Chegou a ter 48% do mercado de browsers em 2009, mas agora está com apenas 3%, segundo estatísticas da Wikipedia [3]. Outros navegadores gratuitos, como Konqueror (nativo de muitas distribuições Linux) e Opera são menos difundidos. Muitas empresas que surgiram voltadas exclusivamente para a Web, como Google e Yahoo, tornaram-se gigantes no mercado de serviços de busca na Web.

Por último, a WWW é responsável por um percentual considerável das vendas de novos computadores, notebooks e tablets. Essa demanda por máquinas para acesso à WWW é justificada em parte pelo aumento da capacidade das conexões e sofisticação da forma como a informação é apresentada (áudio e vídeo em tempo real, principalmente).

## 3 Arquitetura da World-Wide Web

A WWW pode ser considerada como um serviço de aplicação da arquitetura TCP/IP (Transfer Control Protocol/Internet Protocol). Como tal, a arquitetura da WWW:

- segue o modelo cliente/servidor;
- define um esquema de endereçamento na camada de aplicação do modelo OSI/ISO[4];
- define protocolo na camada de aplicação;
- define protocolos nas camadas de apresentação e sessão.

Um servidor Web armazena recursos estruturados segundo um padrão de representação. Servidores são implementados como processos em *background* recebendo requisições via rede no *port*[5] número 80 (default). Clientes acessam estes recursos com o auxílio de um navegador que implementa o lado cliente do protocolo de aplicação e converte uma representação textual em representação gráfica. A Fig. 1 ilustra esta estruturação.

[3] https://en.wikipedia.org/wiki/Usage_share_of_web_browsers

[4] O modelo OSI (Open System Interconnection) padronizado pela ISO (International Organization for Standardization) agrega todas as funções de uma rede de computadores em sete camadas: física (mais baixa), enlace, rede, transporte, sessão apresentação e aplicação (mais alta).

[5] Um *port* (porto ou porta) de rede define, juntamente com o endereço do nó, o endereço de um servidor.

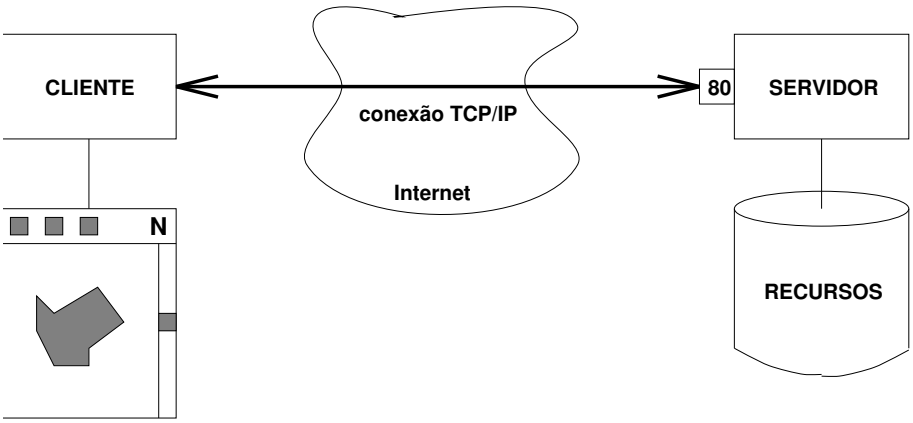


Figure 1: Interação cliente/servidor na WWW.

## 4 Endereçamento na World-Wide Web

O endereçamento no nível de aplicação da arquitetura WWW foi inicialmente especificado pela RFC[6] 1738 (Uniform Resource Locator — URL), o qual foi substituído pela RFC 4248. Um URL é uma sequência de caracteres que identifica de forma global e precisa um recurso na WWW. Genericamente um URL possui a seguinte forma:

```
<esquema>:<parte-especifica-do-esquema>
```

Esquemas especificam recursos (ou serviços) em função dos protocolos utilizados para manipulá-los. Exemplos de esquemas:

- ftp (File Transfer Protocol);
- telnet (protocolo TELNET);
- mailto (protocolo SMTP);
- nntp (Network News Transfer Protocol);
- http (Hypertext Transfer Protocol);
- gopher (protocolo Gopher).

No domínio Internet a parte específica possui a seguinte sintaxe:

```
//user:password@host:port/url-path
```

onde:

- *user* especifica a identificação do usuário;
- *password* designa um *password* de acesso a um recurso (*login* remoto, por exemplo);

[6]Request For Comment: documento de padronização do IETF (Internet Engineering Task Force - http://www.ietf.org/standards/rfcs).

- *host* designa a localização (nó da rede) do servidor que abriga o recurso (notação decimal ou simbólica);
- *port* estipula o número do *port* de rede onde o servidor naquele host aceita conexões;
- *url-path* complementa o URL e é função do esquema.

*User* e *password* são empregados em alguns esquemas onde autenticação é importante (telnet e ftp, por exemplo). Em geral *port* é omitido significando que o port *default* para aquele serviço (recurso) deve ser considerado (21 para ftp, 23 para telnet etc.). *Host* é sempre empregado, apesar de sua ausência ter o significado de "este host".

Vamos examinar os esquemas ftp, telnet, mailto e HTTP para fins de ilustração.

**Esquema ftp**

Com este esquema é possível o acesso a arquivos (isto é, os recursos são arquivos e a operação permitida para este recurso é o acesso). Caso *user* e *password* sejam fornecidos os mesmos são utilizados respectivamente nas diretivas USER e PASS do protocolo FTP. Caso *user* seja omitido o valor "anonymous" é assumido. Caso *user* seja suprido sem *password* um pedido de password é submetido ao usuário. Para os atributos *host* e *port* valem os valores *default* descritos acima.

A parte *url-path* é composta de uma sequência do tipo

```
<cwd_1>/<cwd_2>/.../<cwd_N>/<file>;type=<mode>
```

Para cada argumento <cwd_i> é efetuada uma diretiva CWD <cwd_i> (mude o diretório corrente para <cwd_i>). Após o posicionamento no diretório <cwd_N>, é processada uma diretiva RETR (retrieve) no arquivo <file>. O modo de transferência <mode>, quando presente, significa:

- binário, caso <mode> seja "i";
- US-ASCII, caso <mode> seja "a";
- diretório, caso <mode> seja "d" (neste caso o retorno é o conteúdo do diretório <file>).

Exemplo:

```
ftp://ftp.dca.fee.unicamp.br/pub/docs/prof1/exemplo.exe;type=i
ftp://ftp.dca.fee.unicamp.br/pub/docs/prof2/apostilas/c.pdf;type=a
```

**Esquema telnet**

Neste esquema o recurso é uma sessão interativa de terminal remoto. Apenas o componente *host* do URL é levado em conta. *User* e *password* são ignorados posto que o protocolo TELNET exige autenticação explícita por parte do próprio usuário. *Port* usualmente é omitido sendo utilizado o valor *default* 23. Exemplo:

```
telnet://ssh.fee.unicamp.br/
```

Esse comando não deve mais ser usado para abertura de sessão interativa, já que ele não protege a senha do usuário, a qual é transmitida pela rede sem proteção criptográfica.

**Esquema mailto**

No esquema mailto o recurso é o sistema de correio eletrônio da Internet (protocolo SMTP). Neste esquema *user* e *host* formam o endereço eletrônico Internet. Os demais componentes do URL são desnecessários. Exemplo:

```
mailto://joao@fee.unicamp.br/
```

**Esquema http**

No esquema http os recursos são documentos nos formatos suportados pela WWW (tipicamente HTML). O componente *host* do URL é comumente suprido, sendo *port* optativo (valor *default* é 80). *User* e *password* não são considerados neste esquema.

A parte *url-path* é composta de um caminho para o documento seguido e, opcionalmente, de uma instrução de busca (*query string*) separada do caminho pelo caractere "?". *Query strings* são descritos no contexto da extensão CGI[7].

Exemplo:

```
http://www.fee.unicamp.br/arvore-web/Welcome.html
```

No exemplo acima, o recurso referenciado é o arquivo Welcome.html dentro do diretório "arvore-web" no servidor www.fee.unicamp.br.

# 5 O Protocolo HTTP

HTTP (Hypertext Transfer Protocol) é o protocolo no nível de aplicação da WWW. Este protocolo utiliza o serviço de transporte confiável da Internet (protocolo TCP/IP) e possui uma estrutura tipo *requisição-resposta*. HTTP é especificado pela RFC 2616 (substituída pela RFC 9110) e se encontra na versão 1.1. HTTP difere dos protocolos clássicos de rede (TCP, IP etc.), nos quais o cabeçalho é rígido. Ao contrário, HTTP permite um cabeçalho flexível composto de *linhas* de requisição ou resposta. Linhas são sequências de caracteres no formato US-ASCII (texto mais alguns caracteres especiais como TAB, CR etc.).

HTTP é um protocolo complexo em termos da variedade de requisições/respostas que permite. Entretanto, com exceção da requisição GET,sua variante HEAD e da requisição POST, as demais são pouco usadas. Exemplo de requisições iniciadas por clientes:

- GET: solicita ao servidor o envio de um recurso;
- HEAD: solicita ao servidor o envio de informações sobre um recurso;
- OPTIONS: solicita o envio de informações sobre capacidades do servidor ou recurso;
- TRACE: permite testar a comunicação cliente-servidor;
- POST: permite ao cliente submeter mensagens para listas de distribuição, *newsgroups* etc., bem como retornar o resultado do preenchimento de um formulário;
- PUT: permite ao cliente armazenar ou alterar o conteúdo de um recurso mantido pelo servidor (normalmente desabilitada);

[7] Common Gateway Interface.

- DELETE: solicita a remoção de um recurso mantido pelo servidor (normalmente desabilitada).

As requisições POST, PUT e DELETE dependem de autenticação e verificação de permissões.

Cada requisição e resposta poderá vir acompanhada de *parâmetros*. Parâmetros em HTTP são representados por pares atributo-valor(es). Exemplos de parâmetros:

- HTTP-Version: versão do protocolo (1.0 ou 1.1);
- Date: data GMT da requisição ou resposta;
- Server: indicador do software do servidor;
- Connection: estado da conexão de transporte entre cliente e servidor após o envio da requisição ou resposta;
- Allow: operações HTTP permitidas pelo servidor;
- Host: computador a que se destina a requisição ou resposta;
- User-Agent: indicador do software do cliente e, eventualmente, sistema operacional sobre o qual executa;
- Accept: lista de formatos que o cliente está apto a processar;
- Accept-Encoding: formatos de codificação aceitáveis pelo cliente;
- Accept-Language: linguagens aceitáveis pelo cliente;
- Accept-Charset: formato de caracteres aceitáveis pelo cliente;
- Content-type: indicativo de formato do conteúdo da mensagem HTTP;
- Content-length: tamanho em bytes do conteúdo da mensagem HTTP.

O protocolo HTTP opera da seguinte maneira. O cliente estabelece uma conexão TCP/IP com um servidor executando em determinado host, fica observando determinado *port* (80, caso não fornecido) e aguardando um pedido de conexão. Estabelecida uma conexão com um cliente, este envia ao servidor uma requisição composta da operação, argumentos e parâmetros (requisições são finalizadas por uma linha em branco).

Recebida uma requisição, o servidor a processa, gera uma mensagem de resposta, envia-a pela mesma conexão e encerra esta conexão. Para acessar um recurso, o cliente pode ter que emitir várias requisições. A razão disto é que recursos são transmitidos em partes. Por exemplo, quando um cliente acessa um documento, o servidor o envia sem as figuras. Caso o cliente deseje o documento completo, uma requisição por figura deverá ser efetuada. Esta estratégia visa a recuperação progressiva ou parcial[8] de documentos visando aliviar a carga em conexões de baixa capacidade.

Exemplos de requisições:

[8] Por exemplo, sem as figuras.

```
(a)
GET http://www.fee.unicamp.br/

(b)
GET http://www.fee.unicamp.br/ HTTP/1.1
Host: www.fee.unicamp.br
User-Agent: Mozilla/5.0


(c)
OPTIONS / HTTP/1.1
Host: www.fee.unicamp.br


(d)
TRACE / HTTP/1.1
Host: www.fee.unicamp.br
```

No exemplo (a) o cliente solicita ao servidor o envio do recurso de mais alto nível (home page) do servidor localizado no host www.fee.unicamp.br. A versão do protocolo HTTP e parâmetros da requisição são omitidos.

O exemplo (b) é idêntico ao exemplo (a), exceto que o cliente supre sua versão do protocolo, o host para o qual a requisição se destina e sua identificação.

No exemplo (c) o cliente solicita informações sobre as capacidades do servidor.

Finalmente, no exemplo (d) o cliente testa a comunicação com o servidor. Exemplos de respostas retornadas para os casos acima:

```
(a)
<HTML>
<HEAD>
<title> FEEC Home Page </title>
</HEAD>
.....
</HTML>


(b)
HTTP/1.1 200 OK
Date: Fri, 19 Jun 2015 18:12:32 GMT
Server: Apache/1.3.27 (Unix) PHP/4.2.3 mod_ssl/2.8.12 OpenSSL/0.9.6h
Last-Modified: Tue, 03 Fev 2015 16:32:42 GMT
ETag: "f8901-106-3dc7f4ba"
Accept-Ranges: bytes
Content-Length: 262
Content-Type: text/html


<HTML>
```

```
<HEAD>
<title> FEEC Home Page </title>
</HEAD>
.....
</HTML>
```

```
(c)
HTTP/1.1 200 OK
Date: Fri, 19 Jun 2015 18:22:43 GMT
Server: Apache/1.3.33 (Unix)
Content-Length: 0
Allow: GET, HEAD, OPTIONS, TRACE
```

```
(d)
HTTP/1.1 200 OK
Date: Fri, 19 Jun 2015 18:31:54 GMT
Server: Apache/1.3.33 (Unix)
Transfer-Encoding: chunked
Content-Type: message/http

2e
TRACE / HTTP/1.1
Host: www.fee.unicamp.br
```

## 6 Protocolos WWW de Apresentação e Sessão

No nível de apresentação, a arquitetura da WWW emprega vários protocolos. O protocolo HTML (Hypertext Markup Language) é o protocolo básico de apresentação e especifica uma notação para a composição de documentos hipertexto/hipermídia. Há ainda outros padrões de apresentação. Por exemplo, VRML (Virtual Reality Modeling Language), Java e XML (eXtended Markup Language). Uma vasta gama de padrões de apresentação auxiliares são empregados na composição de documentos. São exemplo de tais padrões: GIF, JPEG (imagens); AVI, MPEG (vídeo) e WAV, AU (áudio).

No nível de sessão é utilizado o mecanismo de *sockets* para o estabelecimento e encerramento de conexões TCP/IP bem como para o envio e recepção de dados através destas conexões. A Fig. 2 ilustra a arquitetura da WWW de acordo com o modelo OSI/ISO.

## 7 Servidores Web

Servidores Web são programas que implementam o protocolo HTTP do lado servidor (isto é, a parte que processa as requisições e gera respostas). Há vários servidores Web disponibilizados em domínio público como, por exemplo:

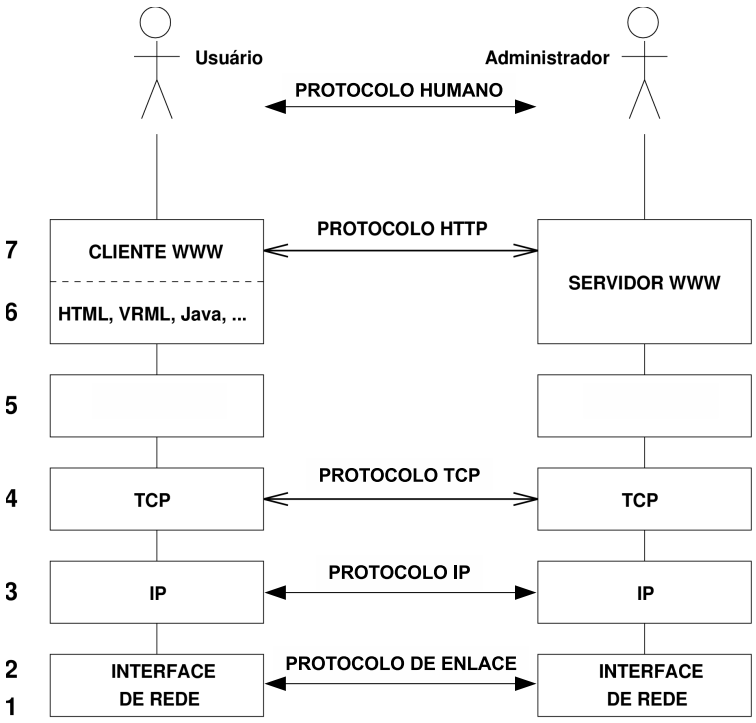


Figure 2: Arquitetura da WWW e o modelo OSI.

1. servidor Apache da Apache Software Foundation;
2. servidor Tomcat da Apache Software Foundation;
3. servidor JBoss (baseado no Tomcat) da JBoss Foundation.

Um servidor típico é apresentado na Fig. 3 e ele possui algumas funcionalidades básicas:

- organizar e gerenciar os recursos a serem disponibilizados na WWW;
- prover acesso seguro à estes recursos;
- prover mecanismos de indexação e busca de recursos;
- processar comandos HTTP e *scripts* especiais (como scripts java, por exemplo).

Via de regra o servidor Web armazena os recursos no próprio subsistema de arquivos do sistema operacional. Adicionalmente, bases de dados e outros dispositivos de armazenamento poderão ser igualmente utilizados.

Servidores Web permitem a incorporação de *scripts* que são executados quando determinadas operações são chamadas pelos clientes (por exemplo, via operação POST) ou quando são explicitamente referenciados. *Scripts* permitem por exemplo que um formulário preenchido eletronicamente seja armazenado num arquivo ou numa base de dados. Interfaces para bases de dados acessadas

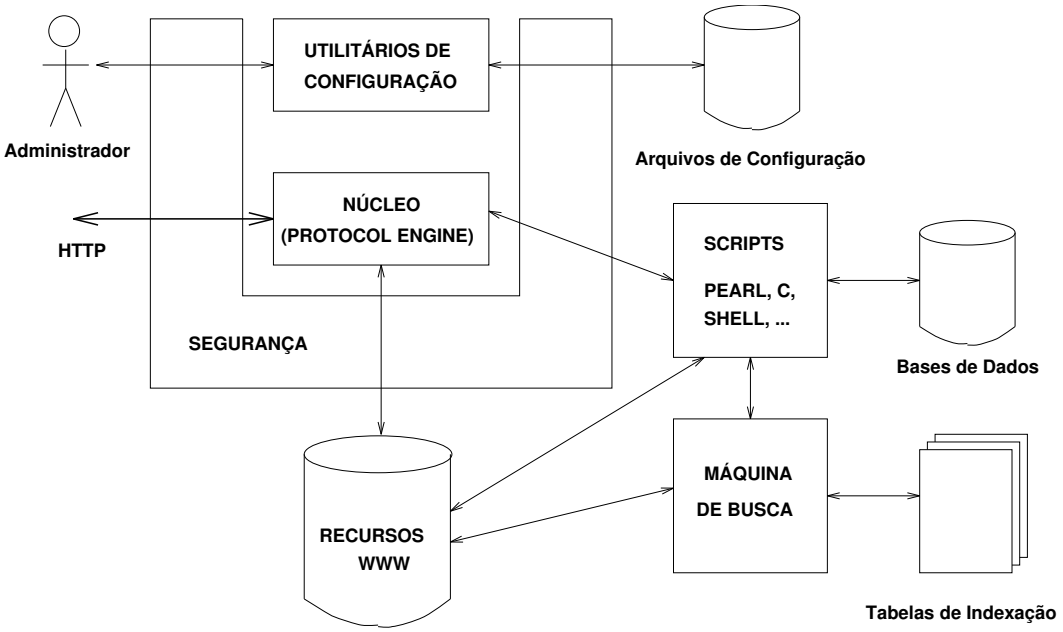


Figure 3: Arquitetura típica de um servidor Web.

por um navegador também são implementadas como *scripts*. Outra utilização de *scripts* é a geração automática de recursos, por exemplo, o resultado de uma consulta a uma base de dados.

Os mecanismos de indexação e busca (*search engines*) permitem que o servidor busque recursos com determinadas características. A busca se dá tipicamente por palavras-chave que podem ocorrer no título do recurso ou em seu conteúdo.

Segurança é sem dúvida uma funcionalidade importante num servidor Web. Usualmente vários níveis de segurança são oferecidos, bem como vários mecanismos de autenticação de requisições. A cobertura da proteção possui tipicamente três níveis:

1. todos os recursos (o servidor inteiro);
2. um subconjunto dos recursos (recursos localizados a partir de um subdiretório);
3. recursos específicos (arquivos individuais).

Quanto aos mecanismos de restrição de acesso, um servidor pode utilizar arquivos internos contendo informações de permissão, ou utilizar um mecanismo externo de autenticação como o NIS[9] ou Kerberos[10].

O escopo da autenticação cobre:

1. usuários individuais, onde cada usuário deve fornecer uma senha de acesso;
2. grupo de usuários, onde é verificado se o usuário acessando o recurso pertence a um grupo autorizado (grupos podem ser formados por usuários individuais ou por grupos de usuários que possuem acesso a determinados recursos em comum).

[9]Network Information System, sistema de autenticação de acesso à rede.
[10]Sistema distribuído de autenticação de elevada segurança desenvolvido pelo MIT.

Arquivos contendo informação de proteção podem ter um carater global (protegem todos os recursos) ou local (protegem os recursos localizados no diretório em que se encontram). Estes arquivos de proteção também são conhecidos como Listas de Controle de Acesso (ACL: Access Control Lists). A administração de servidores Web é tarefa que exige constante atenção por parte do administrador dado o dinamismo com que os recursos são criados, removidos e alterados.

# 8 Navegadores Web

Navegadores são interfaces gráficas concebidas inicialmente para a visualização de arquivos HTML. Entretanto, com o refinamento destas interfaces os navegadores são considerados hoje uma porta de entrada uniforme para muitos serviços da Internet. Mais genericamente, podemos considerar os navegadores como ferramentas de acesso aos recursos Web tais como documentos, arquivos, sessões de acesso remoto e bases de informação, entre outros.

Atualmente, para as principais plataformas existe uma grande variedade de navegadores como, por exemplo:

- Chrome;
- Firefox;
- Safari;
- Edge;
- Opera

entre vários outros menos populares.

Navegadores implementam o lado cliente dos protocolos de aplicação Internet (HTTP, TELNET, FTP, SMTP, NNTP etc.). O tipo de serviço (e consequentemente o protocolo de aplicação) é identificado pelo esquema do URL (http://, ftp:// etc.). Por exemplo, para abrirmos uma sessão de terminal remoto com a máquina azaleia.fee.unicamp.br podemos executar o aplicativo *telnet* em linha de comando ou submeter o URL `telnet://azaleia.fee.unicamp.br` ao navegador.

Um esquema de navegador é apresentado na Fig. 4. A interface gráfica permite ao usuário comandar e configurar o navegador. O núcleo do navegador é responsável por interpretar as ações do usuário. Este componente implementa o lado cliente dos protocolos de aplicação TCP/IP.

Navegadores necessitam de componentes extras de apresentação (que podem até estar embutidos no próprio navegador), tais como:

- tocadores de áudio e vídeo (MPEG, WAV etc.);
- visualizadores de imagens (JPEG, TIFF, GIF etc.);
- visualizadores tridimensionais (VRML etc.);
- formatadores de documentos (PDF, PostScript, LaTeX etc.);
- compactadores/descompactadores (zip, pkzip, gzip etc.);
- interpretadores (Java, TCL/TK etc.);
- utilitários específicos para as mais variadas áreas (comércio eletrônico, bancos etc.).

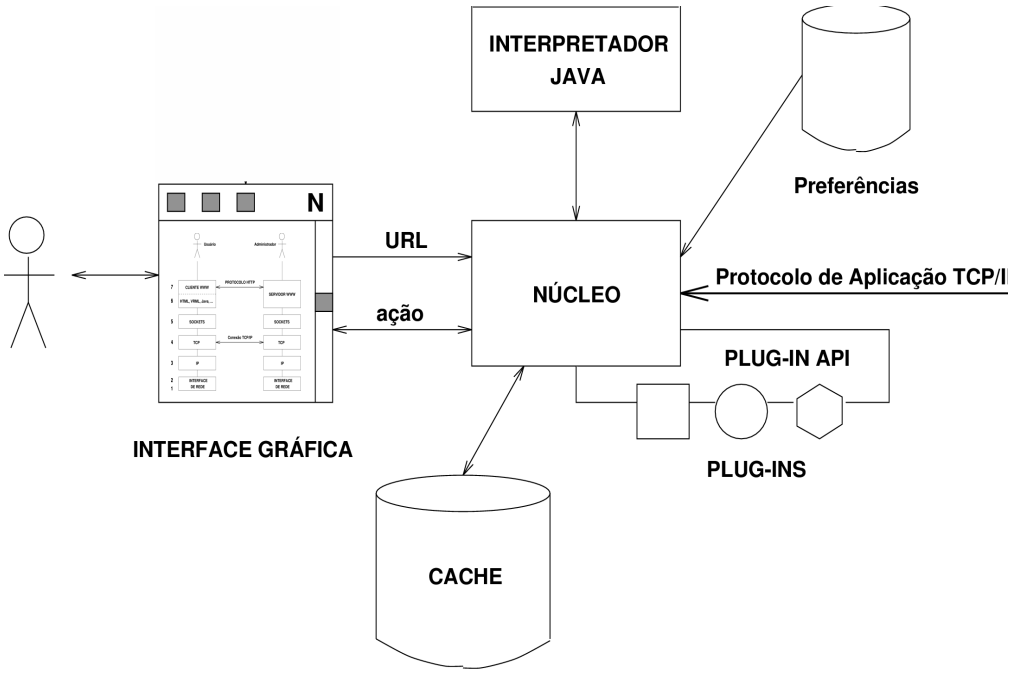


Figure 4: Arquitetura típica de um navegador Web.

Alguns componentes externos são adicionados ao visualizador na forma de arquivos executáveis ou *plug-ins*. O conceito de *plug-in* foi introduzido pela Netscape na versão 2.0 de seu navegador. *Plug-in* é uma interface de programação (API: Application Programming Interface) padronizada responsável por prover uma forma adequada de comunicação entre o navegador e seus componentes externos. No caso do navegador Netscape e seus sucessores Mozilla e Firefox, componentes externos adicionados como *plug-ins* são programas carregados na memória dinamicamente (sob demanda).

Navegadores se utilizam de memória cache para otimizar o acesso a documentos. A idéia básica é manter os documentos acessados recentemente numa memória local (disco) evitando o uso da rede caso estes documentos necessitem ser novamente acessados. Existem duas formas de cache:

1. cache local: subdiretório do usuário onde são armazenados os documentos acessados recentemente pelo usuário;

2. cache global (ou *proxy*): repositório mantido em um ponto da rede[11] onde são armazenados os documentos acessados recentemente pelos usuários desta mesma rede.

# 9 Atividades Práticas

Para realizar os exercícios propostos, o aluno deverá se basear na documentação fornecida. A RFC de número 9110 descreve a versão 1.1 do protocolo HTTP, que será o texto básico para este estudo. Esta RFC substitui a RFC 2616 (já obsoleta). Com um entendimento básico do

[11] Usualmente um por domínio.

protocolo HTTP, o aluno irá conduzir alguns experimentos com o mesmo, objetivando inspecionar as mensagens trocadas entre clientes e servidores HTTP e se familiarizar com a dinâmica destas trocas de mensagens. As atividades estão divididas em duas partes: a primeira para ser feita e entregue até o final da aula e a segunda para ser entregue antes do ínicio da próxima aula.

## 9.1 Primeira parte: atividades para entrega no final da aula

### 9.1.1 Mensagens do navegador

Para registrar o que for feito nesta primeira parte, documente as respostas em um arquivo de texto simples (para submeter ao Moodle daqui a pouco) ou já digite suas respostas diretamente no Moodle (no espaço reservado para esta primeira parte da atividade), tendo o cuidado de especificar bem claramente qual exercício está respondendo.

Primeiramente examinaremos as mensagens de requisição HTTP tipo GET geradas pelos navegadores. Para tal, os seguintes passos devem ser seguidos.

- Compile o programa `http-dump.c` (fornecido junto com este roteiro) em uma máquina Unix/Linux

  ```
  > gcc -o http-dump http-dump.c -lsocket
  ```

  NOTAS:

  - O comando gcc chama o compilador de linguagem C conhecido como GNU C Compiler. É um dos principais produtos de software livre do projeto GNU (significado: GNU is Not Unix).
  - O parâmetro -o indica o nome do arquivo executável a ser produzido pelo compilador.
  - Em certas distribuições de Linux o parâmetro de ligação com a biblioteca de códigos de socket `-lsocket` pode ser omitido.

- Execute o programa `http-dump` passando como parâmetro um *port* maior que 2000, por exemplo:

  ```
  > ./http-dump 5544
  ```

- Usando um navegador comercial (Firefox ou similar), forneça o URL do "servidor" `http-dump`, incluindo o *port* escolhido.

  ```
  http://localhost:5544
  ```

1. (1 pt) Explique em linhas gerais o que faz o programa http-dump que foi compilado, desde que é iniciado até que termina. Não é preciso explicar linha a linha, mas descreva o que fazem os principais trechos deste programa.

2. (1 pt) Documente os valores assumidos pelos parâmetros das mensagens impressas por `http-dump` e explique o significado de cada um dos seguintes elementos.

   (a) GET
   (b) Accept
   (c) Accept-Encoding
   (d) Accept-Language
   (e) Cache-control

(f) Connection

(g) Host

(h) User-Agent

### 9.1.2 Mensagens do servidor

Agora vamos examinar as respostas geradas por servidores HTTP às requisições enviadas pelos navegadores. Como navegadores processam o protocolo HTTP de forma transparente ao usuário utilizaremos um recurso alternativo para poder ver as respostas de maneira explícita.

1. (1 pt) Faça um `telnet` para uma máquina onde exista um servidor HTTP real instalado, passando o *port* do servidor (80).

   `telnet www.fee.unicamp.br 80` (ou outro servidor como `www.example.org` ou `www.gov.br`)

   (a) (0,5 pts) Explique por que o *port* 80 é necessário e o que acontece se ele não for fornecido.

   (b) (0,5 pts) O que ocorreria se o *port* fornecido fosse o de número 443? Mesmo que não funcione em seus testes, procure descobrir e relate o que este *port* tem de especial.

2. (1 pt) Digite os comandos HTTP a seguir (terminando-os com uma linha em branco, isto é, com duas inserções da tecla `<ENTER>`), e documente o que foi retornado como resposta. (Obs: como o servidor pode ter um tempo de tolerância curto, pode ser mais prático copiar e colar no terminal as duas linhas de cada comando.)

   (a) (0,1 pts.) Envie o comando e documente o resultado.

   `GET / HTTP/1.1`

   `Host: www.fee.unicamp.br` (ou o nome de outro sítio web escolhido)

   (b) (0,3 pts.) Envie o comando, documente o resultado e explique a diferença do resultado de HEAD com o de GET.

   `HEAD / HTTP/1.1`

   `Host: www.fee.unicamp.br` (ou o nome de outro sítio web escolhido)

   (c) (0,3 pts.) Envie o comando, documente o resultado e explique a diferença do resultado de GET /index.html HTTP/1.1 com o de GET / HTTP/1.1 .

   `GET /index.html HTTP/1.1`

   `Host: www.fee.unicamp.br` (ou o nome de outro sítio web escolhido)

   (d) (0,3 pts.) Envie o comando, documente o resultado e explique a diferença do resultado de GET HTTP/1.1 com o de GET / HTTP/1.1 .

   `GET HTTP/1.1`

   `Host: www.fee.unicamp.br` (ou o nome de outro sítio web escolhido)

## 9.2 Segunda parte: atividades para entrega posterior

1. (1 pt) Faça um `telnet` para uma máquina onde exista um servidor HTTP real instalado, passando o *port* do servidor (80). Exemplo para o sítio web da FEEC, mas pode ser outro, como `www.example.org` ou `www.gov.br`.

   `telnet www.fee.unicamp.br 80` (ou outro sítio web escolhido)

Digite os comandos HTTP a seguir (terminando-os com uma linha em branco, isto é, com duas inserções da tecla `<ENTER>`), e documente o que foi retornado como resposta. (Obs: como o servidor pode ter um tempo de tolerância curto, pode ser mais prático copiar e colar no terminal as duas linhas de cada comando.)

(a) (0,2 pts) Envie o comando, documente o resultado e explique a diferença entre o resultado de OPTIONS e o de HEAD feito anteriormente no mesmo servidor.

```
OPTIONS / HTTP/1.1
Host:  www.fee.unicamp.br (ou o nome de outro sítio web escolhido)
```

(b) (0,2 pts) Envie o comando, documente o resultado e explique se houve (e qual foi a) diferença entre o resultado esperado e o obtido.

```
TRACE / HTTP/1.1
Host:  www.fee.unicamp.br (ou o nome de outro sítio web escolhido)
```

(c) (0,2 pts) Envie o comando, documente o resultado e explique se houve (e qual foi a) diferença entre o resultado esperado e o obtido.

```
POST / HTTP/1.1
Host:  www.fee.unicamp.br (ou o nome de outro sítio web escolhido)
```

(d) (0,2 pts) Envie o comando, documente o resultado e explique se houve (e qual foi a) diferença entre o resultado esperado e o obtido.

```
DELETE / HTTP/1.1
Host:  www.fee.unicamp.br (ou o nome de outro sítio web escolhido)
```

(e) (0,2 pts) Envie o comando, documente o resultado e explique se houve (e qual foi a) diferença entre o resultado esperado e o obtido.

```
AGORA_VAI / HTTP/1.1
Host:  www.fee.unicamp.br (ou o nome de outro sítio web escolhido)
```

2. (0,5 pts) Explique o que ocorre se for usado na linha `Host` do exercício anterior um endereço diferente daquele indicado no comando `telnet` inicial. Faça testes com pelo menos dois servidores diferentes (pelo menos um fora da Unicamp) e os documente para comprovar sua resposta.

3. (1 pt) Explique o significado dos termos abaixo, consultando as RFCs indicadas ou outros documentos na web.

   (a) 200 OK
   (b) 400 Bad Request
   (c) 401 Unauthorized
   (d) 403 Forbiden
   (e) 404 Not Found
   (f) 405 Method Not Alowed
   (g) 500 Internal Server Error
   (h) 501 Not Implemented
   (i) Accept-Ranges

(j) Allow
(k) ETag
(l) X-Pad
(m) Content-Length
(n) Content-Type
(o) Date
(p) Last-Modified
(q) Server
(r) Transfer-Encoding
(s) WWW-Authenticate
(t) Chunked

4. (0,5 pts) Crie e apresente testes de troca de mensagens com um ou mais servidores web usando `telnet`, testes estes que resultem em pelo menos 5 códigos de erro HTTP distintos.

5. (1 pt) Usando um navegador web, visualize o conteúdo de `www.fee.unicamp.br` ou de outro sítio web que tenha desviado o acesso da porta 80 (protocolo HTTP) para outra (443 do protocolo HTTPS), como `www.gov.br`, por exemplo. Veja os códigos que originaram esta página inicial, solicitando ao seu navegador que mostre os códigos-fonte da mesma (há um menu específico para isso nos navegadores), e veja que se trata de um arquivo relativamente grande com muitos rótulos (tags) em HTML que dão o formato final da página inicial vista. De forma totalmente diferente, o comando `GET /` submetido ao servidor via `telnet` resulta em um pequeno recurso (arquivo) no padrão HTML.

   (a) (0,3 pts) Explique esta diferença e o que este pequeno conteúdo obtido via `telnet` está indicando para o navegador.
   (b) (0,7 pts) Com base nos exercícios anteriores, monte um exemplo usando o comando `telnet` e mostre/explique como o navegador consegue chegar à página web definitiva do sítio web. Documente os comandos usados e os conteúdos retornados. No caso da página definitiva (grande), não é preciso documentar (copiar no relatório) todo o código-fonte da mesma, mas apenas algumas linhas no início e fim.

6. (2 pts) Redirecione a saída de `http-dump` para um arquivo de saída qualquer e acesse este "servidor" com dois navegadores de fornecedores diferentes (Firefox, Edge, Chrome, Opera, Safari etc). Em seguida adapte o programa criado no exercício 5 da Atividade 3 para guardar nas listas especificadas naquele exercício os diversos nomes de campos e seus valores presentes em requisições HTTP. Observe que a linha inicial que traz o comando HTTP (GET ou TRACE ou PUT ou POST ou OPTIONS) não tem o símbolo de dois pontos (:) e precisará de tratamento especial para separar e guardar seus parâmetros que vêm logo depois. Após a linha de comando HTTP temos várias outras linhas com nomes de campos, antes da pontuação ":", e com os valores desses campos após os dois pontos e separados por vírgulas. É preciso atentar também para linhas que tenham o separador (:) mais de uma vez (por causa da especificação de número de porta). Além de documentar seu código de armazenamento em listas no relatório, mostre que o mesmo está funcionando corretamente, mandando-o varrer as listas e imprimir na tela todos os dados obtidos das mesmas.