CES-35 – Redes de Computadores e Internet

# Laboratório número 2
# Implementando Cliente e Servidor HTTP

Neste laboratório, o objetivo é ensinar programação de sockets através do design de um protocolo HTTP básico e por meio de desenvolvimento de um servidor e cliente Web simples. Todas as implementações devem ser escritas em C ou C++ usando a biblioteca padrão sockets BSD. Para o laboratório não será permitido bibliotecas de abstração de rede de mais alto nível (como Boost.ASIO ou similares).

O laboratório possui 2 trechos de código a serem desenvolvidos: um arquivo contendo a implementação do servidor Web e um outro arquivo contendo o cliente Web. O servidor Web aceita uma solicitação HTTP, analisa a solicitação e procura o arquivo solicitado dentro do diretório de arquivos local. Se o arquivo solicitado existir, o servidor retornará o conteúdo do arquivo como parte de uma resposta HTTP, caso contrário, deverá retornar a resposta HTTP apropriada, com o código correspondente. O cliente ao recuperar a resposta Web deve salva-la em arquivo no diretório local. A ideia é implementar o equivalente ao HTTP 1.0, onde as conexões não são persistentes, é preciso um socket por objeto.

Para esse projeto só é preciso implementar a solicitação GET com a correspondente resposta.

1. O servidor:
   O servidor Web será executado pela linha de comando, e aceita 3 argumentos:
   nome do host do servidor web, número de porta e nome de diretório onde ficarão os arquivos a serem servidos.

   $ web-server [host] [port] [dir]

   Exemplo: $ ./web-server localhost 3000 /tmp
   onde localhost é um nome usado para se referenciar a própria máquina, escutar na porta 3000 e os arquivos servidos estarão no diretório /tmp.

   Algumas atividades que o servidor web precisa realizar incluem:
   a) converter o nome do host do servidor em endereço IP, abrir socket para escuta neste endereço IP e no número de porta especificado.
   b) por meio do socket "listen" aceitar solicitações de conexão dos clientes, e estabelecer conexões com os clientes.
   c) Fazer uso de programação de redes que lide com conexões simultâneas (por exemplo, por meio de multiprocess e multithreads). Ou seja, o servidor web deve poder receber solicitações de vários clientes ao mesmo tempo.

2. O cliente:
   O cliente será executado pela linha de comando, que aceitará como argumento uma URL.

   $ ./web-client [URL]

   Um possível exemplo, poderia ser a solicitação de um arquivo index.html do servidor web local, na mesma máquina. Usando um comando assim.

$ ./web-client http://localhost:3000/index.html
O cliente Web precisa então:
1) segmentar a string da URL e interpretar os parametros da mesma como hostname, porta e o objeto a ser solicitado.
2) o cliente precisa se conectar por TCP com o servidor Web.
3) Assim que a conexão for estabelecida, o cliente precisa construir uma solicitação HTTP e enviar ao servidor Web e ficar bloqueado aguardando uma resposta.
4) Após receber a resposta, o cliente precisa analisar se houve sucesso ou falha, por meio de análise do código de resposta. Se houver sucesso, ele deve salvar o arquivo correspondente no diretório atual usando o mesmo nome interpretado pela URL.

3. Observações gerais:
    - Lembre-se que uma solicitação GET é delimitada no socket pelo recebimento da linha em branco (em binário "\r\n\r\n").
    - Já a resposta 200 OK é mais envolvida e requer um parsing de um cabeçalho HTTP para descobrir o tamanho do conteúdo e a leitura de N bytes correspondentes a esse tamanho.
    - Os arquivos de teste não devem ser maiores que 1MB.
    - A implementação deve suportar somente os seguintes códigos de status: 200 OK, HTTP 400 Bad Request, 404 Not Found.
    - O mais fácil para realizar o projeto é usar um ambiente de desenvolvimento baseado em Linux padronizado.
    - Os arquivos a serem gerados devem ser web-server.cpp e web-client.cpp.
    - Gere um .tar.gz contendo o fonte e executável.

Você deve testar seu cliente puxando a página do Kurose:
./web-client http://gaia.cs.umass.edu:80/wireshark-labs/HTTP-wireshark-file1.html