LINGUAGEM DE PROGRAMAÇÃO II

Centro de Informática – Universidade Federal da Paraíba – Campus I

Professor: Carlos Eduardo Batista - Semestre: 2018.2

TRABALHO PRÁTICO: *COMUNICAÇÃO INTER-PROCESSOS*

Entrega por e-mail: 03/05/2019.

Arquivo de entrega deve anexar *todos* os códigos fonte em C/C++ dentro de um diretório nomeado "MATRICULA_ALUNO-LP2-Trabalho2" o qual deve ser comprimido em um arquivo ZIP ("MATRICULA_ALUNO-LP2-Trabalho2.zip"). O arquivo ZIP deve conter obrigatoriamente um arquivo de texto chamado README.txt contemplando **todas** as instruções de compilação e execução, além de qualquer observação que se fizer necessária para correção.

Mini-servidor HTTP

O trabalho consiste na implementação de um programa em C/C++ que funcione como um mini-servidor HTTP, que receberá e responderá, através do protocolo HTTP[1] (que é baseado em TCP) requisições do tipo **GET** , que, no seu processamento, recuperarão arquivos localizados em uma pasta específica e os utilizarão para composição de uma resposta a ser enviada para o cliente solicitante. A implementação utilizará a biblioteca de sockets de C/C e também programação concorrente, em lógica detalhada a seguir.

O programa a ser implementado é um mini-servidor HTTP *multithreaded* que deve ser capaz de processar múltiplas requisições simultâneas em concorrência (mínimo de dez requisições). Você deverá demonstrar que seu servidor HTTP é capaz de enviar os arquivos solicitados a um cliente Web (e.g. um browser). O programa deverá conter em seu diretório raiz um diretório chamado ***files***, o qual conterá arquivos sortidos a serem servidos (imagens JPEG e arquivos HTML, por exemplo) e um arquivo chamado ***log.txt*.**

Ao ser iniciado, o programa deverá receber (ou definir por padrão) um valor inteiro a ser utilizado como porta (exemplo: 80 ou 8080) para a criação um *socket* (*StreamSocket*/TCP) servidor, que aguardará requisições dos clientes. O servidor deverá responder solicitações HTTP do tipo **GET**[2], como a exibida a seguir:

```
GET /arquivo.html HTTP/1.1

Host: www.test101.com

Accept: image/gif, image/jpeg, */*

Accept-Language: en-us

Accept-Encoding: gzip, deflate

User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)

(linha em branco)
```

[1] https://tools.ietf.org/html/rfc2616
[2] https://tools.ietf.org/html/rfc2616#page-53

Onde o arquivo chamado **arquivo.html** é solicitado, e outras informações são passadas como cabeçalho de requisição (*Host, Accept, Accept-Language* etc.). O programa implementado deverá tratar em *threads* individuais cada uma das requisições recebidas, e gerar uma resposta de acordo com o que define o padrão HTTP. Na resposta, deverão ser informados: a data e hora que a resposta foi gerada (campo **Date**); o nome do servidor (campo **Server**, contendo uma string arbitrária que identifique seu servidor); o tamanho do arquivo incluído na resposta (campo **Content-Length**, em bytes); o tipo do arquivo incluído na resposta (campo **Content-Type**, seguindo o padrão MIME[3] - pelo menos dois tipos de arquivos devem ser suportados, sugestão: HTML e JPG).

A seguir uma resposta válida gerada para a requisição de exemplo:

```
HTTP/1.1 200 OK

Date: Sun, 18 Oct 2009 08:56:53 GMT

Server: Apache/2.2.14 (Win32)

Content-Length: 44

Content-Type: text/html

Connection: Closed


<html><body><h1>Funciona!</h1></body></html>
```

(conteúdo do arquivo.html)

Deverão ser tratadas duas situações de exceção: uma para quando o arquivo solicitado não existir e outra para quando a requisição for mal formada (i.e. não seguir o padrão definido ou não utilizar o método GET). Para estes casos, respostas como as abaixo deverão ser geradas, respectivamente:

```
HTTP/1.1 404 Not Found
Date: Sun, 18 Oct 2012 10:36:20 GMT
Server: Apache/2.2.14 (Win32)
Content-Length: 230
Content-Type: text/html; charset=iso-8859-1
Connection: Closed
```

```
HTTP/1.1 400 Bad Request
Date: Sun, 18 Oct 2012 10:36:20 GMT
Server: Apache/2.2.14 (Win32)
Content-Length: 230
Content-Type: text/html; charset=iso-8859-1
Connection: Closed
```

[3] http://www.iana.org/assignments/media-types/media-types.xhtml

É importante atentar para a formatação definida pelo protocolo, conforme definido no RFC (quebras de linha separando cabeçalho da requisição e da resposta etc.). Além de responder individualmente cada solicitação, as requisições deverão ser armazenadas em um arquivo de registro, armazenado na pasta raiz do programa e chamado de **log.txt.** Todas as informações recebidas na requisição devem ser armazenadas no arquivo de registro, incluindo: **cabeçalhos e conteúdo da requisição e endereço IP do cliente solicitante**. O acesso de escrita e leitura ao arquivo de registro deve ser protegido pelo uso de travas, monitores e/ou semáforos para garantir que todas as requisições sejam registradas no arquivo de registro pela ordem de chegada.

A implementação deverá ser feita em C/C++ utilizando a biblioteca Pthreads ou as threads STD (C++11). A implementação deverá atender às quatro propriedades de uma solução para o problema da seção crítica: exclusão mútua, ausência de *deadlock*, ausência de atraso desnecessário e entrada eventual. A saída do seu programa deve ser bem planejada, de forma a mostrar o que está acontecendo a cada momento.

A pontuação se dará da seguinte forma:

Criação correta do socket servidor (4,0)

Tratamento adequado das requisições GET (4,0)

Tratamento de erros (2,0)

Implementação de registro (log) com semáforos: (3,0)

Nota máxima: 10,0