Universidade Federal de Pernambuco - UFPE
Centro de Informática – CIn
Curso de Bacharelado em Sistemas de Informação (BSI)

DANIEL FREIRE TURMINA - dft@cin.ufpe.br
YURI CORREIA DE BARROS - ycb@cin.ufpe.br

PROJETO DE DESENVOLVIMENTO COM SOCKETS EM PYTHON

RECIFE, PE
2021

Universidade Federal de Pernambuco - UFPE
Centro de Informática – CIn
Curso de Bacharelado em Sistemas de Informação (BSI)

DANIEL FREIRE TURMINA - dft@cin.ufpe.br
YURI CORREIA DE BARROS - ycb@cin.ufpe.br

PROJETO DE DESENVOLVIMENTO COM SOCKETS EM PYTHON

Relatório do desenvolvimento de projetos baseados na implementação de servidores utilizando-se de sockets e dos protocolos da camada de transporte TCP e UDP, apresentado como parte do requisito para obtenção de nota na disciplina de Redes de computadores.

Orientação: Prof. Kevin Lopes e Prof. Ygor Amaral

RECIFE, PE
2021

SUMÁRIO

1. INTRODUÇÃO

Este relatório tem o objetivo de apresentar os resultados de dois projetos que foram desenvolvidos para a criação de servidores. Esses projetos foram implementados na linguagem Python e estão pautados na utilização de socket, como ferramenta necessária para o estabelecimento de uma comunicação efetiva entre clientes e servidor. Para estabelecer esta comunicação, um projeto utiliza do protocolo da camada de transporte UDP e o outro foi desenvolvido com base no protocolo TCP.

Com o intuito de facilitar a apresentação dos resultados obtidos nos projetos, é importante conceituar e detalhar as diferenças entre os dois protocolos mencionados anteriormente. O TCP é um protocolo que garante a entrega dos dados, oferecendo um transporte confiável, realiza o controle de fluxo e do congestionamento e é orientado a conexão, ou seja, antes de um cliente realizar requisições para um servidor, é necessário estabelecer uma conexão com ele. Já o protocolo UDP não é orientado a conexão, não garante a entrega de dados e não possui controles de fluxos e de congestionamento. Apesar disso, alguns serviços utilizam-se do UDP, pois ele é um protocolo muito mais rápido. Como não é necessário estabelecer conexão prévia, os atrasos são menores e em muitos casos é tolerável uma quantidade pequena de perda de dados (KUROSE; ROSS, 2013).

2. CRONOGRAMA

O cronograma das atividades está listado na Tabela 1.

Atividade	**Data**
Divulgação dos Projetos via Google Classroom	01/07/2021
Início do Projeto 1 – Quiz Competitivo	02/07/2021
Início do Relatório	27/07/2021
Início do Projeto 2 – Servidor Web	04/08/2021
Conclusão do Projeto 1	30/07/2021
Conclusão do Projeto 2	10/08/2021
Conclusão do Relatório	18/08/2021
Entrega e Apresentação dos Projetos	19/08/2021

Tabela 1 - Cronograma de atividades

3. PROJETO 1 – QUIZ COMPETITIVO

3.1 Descrição

No projeto 1 foi desenvolvido um jogo de perguntas e respostas sobre as seleções de futebol que já foram campeãs do mundo. Para isso foram desenvolvidos dois códigos escritos na linguagem Python, um para os clientes, que no caso seriam os usuários (jogadores) e outro para o servidor que seria o responsável por gerenciar toda a competição: cadastro dos jogadores participantes, envio das perguntas, análise das respostas, gerenciamento do tempo e das pontuações e envio da classificação final do jogo.

3.2 Metodologia

Para permitir o envio das requisições e respostas entre os clientes e o servidor foram implementados sockets e foi utilizado o protocolo da camada de transporte UDP. Além disso, um próprio protocolo da camada de aplicação foi desenvolvido para que aplicação do cliente fosse capaz de interpretar as respostas recebidas do servidor, conforme tabela abaixo.

Código	Mensagem
100	'Digite "start" para começar:'
101	'Sua partida está prestes a começar'
102	'A quantidade limite de jogadores foi atingida! Tente novamente mais tarde! '
103	'Infelizmente você não digitou "start" a tempo, a partida já iniciou!'
104	'Partida já iniciada, volte mais tarde!!'
200	'Você acertou!'
300	'Você errou! Tente novamente...'
400	'Uma nova partida está para começar. Deseja iniciar? [s/n]'
500	'Ok aguarde o início do jogo!'
501	'Tudo bem, obrigado por participar, até logo!'

502	'Aguarde até que o jogo reinicie!'
600	'A partida irá reiniciar em instantes'

Tabela 2 – Códigos: Protocolo da camada de aplicação

O código criado para estabelecer o servidor inicia importando algumas bibliotecas que serão necessárias para a criação do programa. A primeira biblioteca é a “socket” e é com esta que se consegue estabelecer o socket do servidor e utilizar o protocolo UDP.

Logo em seguida, algumas estruturas de dados foram criadas: lista para inserir os jogadores que se conectarem, lista para inserir os jogadores que estiverem prontos para a partida, variáveis para controle do início da partida e dos tempos, dicionários para relacionar o nome dos jogadores com seus endereços (IP e porta) e com suas pontuações e uma variável com a porta na qual o servidor estará recebendo as requisições.

O servidor então faz a leitura do arquivo que contém todas as perguntas e respostas registradas e escolhe de forma aleatória 05 conjuntos de perguntas e respostas distintas. A partir disso, várias funções foram implementadas: para receber mensagens e decodificá-las, para enviar mensagens, para contagem dos tempos, para controle dos jogadores e da partida em si.

Nessas três últimas funções foi utilizada a biblioteca “threading”, permitindo que duas ou mais partes do programa fossem executadas de forma paralela, tendo em vista que era necessário atender as requisições de diversos clientes de forma concorrente. Por fim, o servidor realiza a soma final da pontuação obtida por cada jogador, envia para todos a classificação final da partida e pergunta se desejam começar um novo jogo.

Do lado do cliente também foram utilizadas as bibliotecas “socket” para permitir o envio de requisições para o servidor e para receber as respostas e o “threading” para permitir que as funções de enviar e receber mensagens pudessem ser utilizadas simultaneamente.

3.3 Resultados Obtidos

O jogo atendeu a todos os critérios estabelecidos na especificação do projeto. O jogo inicia a partir do momento em que pelo menos 02 jogadores estão prontos para o jogo e tem um limite de até 05 jogadores por competição. Após a competição ser iniciada, não é permitido o ingresso de novos competidores e cada competição tem 05 rodadas de perguntas e respostas. Ao início de cada competição, 05 perguntas distintas são escolhidas de forma aleatória e cada rodada é encerrada quanto algum jogador acerta a resposta ou quando se passam 10 segundos. Ao final é calculado as pontuações de cada jogador, sendo 25 pontos por cada resposta certa,

-5 pontos por cada resposta errada e - 1 ponto por cada rodada sem resposta. Conclui-se apresentando o ranking final e uma nova competição poderá ser iniciada.

Descrição do Processo	Terminal do Servidor	Terminal do Cliente
Mensagem inicial Ao iniciar os processos e no estabelecimento da conexão com os jogadores	Servidor iniciado, perguntas sorteadas e aguardando conexões. Conectado: ‘Nome do Jogador’ (‘IP’, ‘Porta’)	--- Quiz: Campeões do Mundo! --- Digite seu nome:
Preparação para início da competição Aguardando quantidade mínima de jogadores “prontos” para a competição	Jogadores prontos: [“lista”]	Digite "start" para começar:
Início da Competição Com pelo menos dois jogadores prontos, um cronômetro de 10 segundos é iniciado para começar a partida	Iniciando cronômetro... Cronômetro fechado	Sua partida está prestes a começar 10 9 ... 3 2 1
Perguntas O jogo inicia e o servidor começa a mandar as perguntas e um cronômetro de 10 segundos é iniciado para cada rodada	Iniciando cronômetro...	Exemplo: “Quem ganhou a Copa do Mundo de 1978? ”
Acertando a Resposta O servidor recebe a resposta e confere se está correta. O servidor envia que a resposta está certa e encerra o cronômetro.	O jogador: ‘Nome do Jogador’ acertou a pergunta Cronômetro fechado	Você acertou! O jogador: ‘Nome do Jogador’ acertou!
Errando a Resposta O servidor confere a resposta e envia que a resposta está errada.	O jogador: ‘Nome do Jogador’ errou a pergunta	Você errou! Tente novamente...

Tempo Esgotado Ninguém respondeu corretamente em 10 segundos. O Servidor encerra a rodada e envia a próxima pergunta.	Cronômetro fechado	-
Outro jogador acertou a resposta O servidor informa aos outros clientes que um jogador acertou.	Cronômetro fechado	O jogador: ‘Nome do Jogador’ acertou!
Fim da Competição	Cronômetro fechado	1° Lugar: ‘Nome do Jogador’ com ‘X’ pontos! 2° Lugar: ‘Nome do Jogador’ com ‘X’ pontos! Fim de Jogo - Obrigado!!
Jogador tentando se conectar e participar de uma competição já iniciada	-	Partida já iniciada, volte mais tarde!!
Jogador se conectou, mas não digitou start a tempo	-	Infelizmente você não digitou "start" a tempo, a partida já iniciou!
Jogador tentando participar de uma competição com limite de 05 jogadores já atingida	-	'A quantidade limite de jogadores foi atingida! Tente novamente mais tarde! '
Início de uma nova competição Ao fim de uma competição, o servidor questiona se os jogadores desejam começar uma nova partida.	-	Uma nova partida está para começar Deseja iniciar? [s/n]
Jogador deseja participar da nova competição Caso o jogador deseje participar de uma nova competição, ele irá aguardar por outros jogadores.	-	'Ok aguarde o início do jogo!' 'A partida irá reiniciar em instantes'
Jogador não deseja participar da nova competição	-	'Tudo bem, obrigado por participar, até logo!'

Tabela 3 – Terminal do servidor e do cliente em diversas etapas do quiz.

4. PROJETO 2 – SERVIDOR WEB

4.1 Descrição

No projeto 2 foi desenvolvido um Servidor WEB implementando o protocolo da camada de aplicação HTTP e o protocolo da camada de transporte TCP. Para isso foram desenvolvidos dois códigos escritos na linguagem Python, um para o cliente, que no caso seriam os usuários (permitindo simular requisições diferentes de GET e simular o uso de versões diferentes do HTTP) e outro para o servidor que seria o responsável por gerenciar todo servidor web: encaminhar as páginas solicitadas, arquivos, imagens e apresentar mensagens em caso de algum erro encontrado.

4.2 Metodologia

Para permitir o envio das requisições e respostas entre os clientes e o servidor foram implementados sockets e foi utilizado o protocolo da camada de transporte TCP. O código criado para estabelecer o servidor inicia importando algumas bibliotecas que serão necessárias para a criação do programa. A primeira biblioteca é a "socket" e é com esta que se consegue estabelecer o socket do servidor e utilizar o protocolo TCP. Definimos a porta padrão (no nosso projeto foi a 8081) e iniciamos o servidor para escutar as requisições. Assim, quando um cliente tentar acessar o servidor, será realizado o three-way handshake, estabelecendo a conexão cliente-servidor (isso acontece, pois estamos utilizando o TCP).

O three-way handshake acontece no momento em que o cliente solicita a conexão com o servidor (SYN) e caso a resposta seja aceita (SYN-ACK), ele poderá enviar a requisição. Para responder as requisições foi implementado protocolo da camada de aplicação HTTP. O servidor caso tenha recebido uma requisição GET poderá enviar página html, arquivos ou passear pelas pastas, a depender do caso ou então irá enviar um arquivo html personalizado caso tenha ocorrido algum tipo de erro, conforme tabela abaixo:

Código	Descrição
200 OK	Requisição bem-sucedida, objeto requisitado será enviado
400 Bad Request	Mensagem de requisição não entendida pelo servidor (Erro de sintaxe na requisição)
404 Not Found	Documento requisitado não localizado no servidor
501 Not Implemented	Tipo de requisição não implementado no servidor (Diferente de GET)
505 HTTP Version Not Supported	Versão do HTTP utilizada não é suportada neste servidor

Tabela 4 – Códigos: Protocolo da camada de aplicação do Servidor Web

4.3 Resultados Obtidos

Apresentaremos a seguir as respostas do servidor web a depender do tipo e formato da requisição.

4.3.1 Erro 400 Bad Request

Mensagem enviada pelo cliente (recebida pelo servidor web), contendo um erro na sintaxe da requisição:

```
messagem_enviada = 'GET ERRO_SINTAXE HTTP/1.1\r\n'
messagem_enviada += 'Host: localhost\r\n'
messagem_enviada += '\r\n'
```

Figura 1 – Requisição com erro na sintaxe

Mensagem recebida pelo cliente (enviada pelo servidor), contendo um arquivo html informando que a mensagem não foi compreendida pelo servidor:

```
HTTP/1.1 400 Bad Request
Date: Wed Aug 18 16:09:16 2021
Server: YD-Server Win 10
Content-Type: text/html


                    <!DOCTYPE html>
                    <html lang="en">
                    <head>
                        <meta charset="UTF-8">
                        <title>400 Bad Request</title>
                    </head>
                    <body>
                        <h1>400 Bad Request</h1>
                        <h2>Mensagem de requisição não entendida pelo servidor<h2>
                    </body>
                    </html>
```

Figura 2 – Erro 400 Bad Request

4.3.2 Erro 404 Not Found

Mensagem enviada pelo cliente (recebida pelo servidor web), requisitando um arquivo que não existe no servidor:

```
messagem_enviada = 'GET /aquivoNaoExistente.txt HTTP/1.1\r\n'
messagem_enviada += 'Host: localhost\r\n'
messagem_enviada += '\r\n'
```

Figura 3 – Requisição de arquivo inexistente

Mensagem recebida pelo cliente (enviada pelo servidor), contendo um arquivo html informando que este arquivo não foi localizado no servidor:

```
HTTP/1.1 404 Not Found
Date: Wed Aug 18 16:27:48 2021
Server: YD-Server Win 10
Content-Type: text/html


                        <!DOCTYPE html>
                        <html lang="en">
                        <head>
                            <meta charset="UTF-8">
                            <title>404 Not Found</title>
                        </head>
                        <body>
                            <h1>404 Not Found</h1>
                            <h2>Documento requisitado não localizado no servidor<h2>
                        </body>
                        </html>
```

Figura 4 – Erro 404 Not Found

O erro 404 Not Found também foi testado no navegador, conforme imagem abaixo:

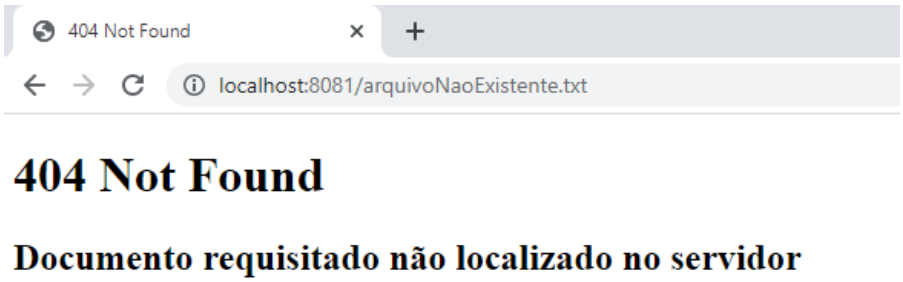

Figura 5 – Erro 404 Not Found no navegador

4.3.3 Erro 501 Not Implemented

Mensagem enviada pelo cliente (recebida pelo servidor web), contendo uma requisição do tipo POST:

```
messagem_enviada = 'POST / HTTP/1.1\r\n'
messagem_enviada += 'Host: localhost\r\n'
messagem_enviada += '\r\n'
```

Figura 6 – Requisição do tipo POST

Mensagem recebida pelo cliente (enviada pelo servidor), contendo um arquivo html informando que este tipo de requisição não foi implementado no servidor:

```
HTTP/1.1 501 Not Implemented
Date: Wed Aug 18 16:14:35 2021
Server: YD-Server Win 10
Content-Type: text/html


                    <!DOCTYPE html>
                    <html lang="en">
                    <head>
                        <meta charset="UTF-8">
                        <title>501 Not Implemented</title>
                    </head>
                    <body>
                        <h1>501 Not Implemented</h1>
                        <h2>Tipo de requisição não implementado no servidor<h2>
                    </body>
                    </html>
```

Figura 7 – Erro 501 Not Implemented

4.3.4 Erro 505 HTTP Version Not Supported

Mensagem enviada pelo cliente (recebida pelo servidor web), contendo uma requisição que utiliza a versão 2.0 do HTTP:

```
messagem_enviada = 'GET / HTTP/2.0\r\n'
messagem_enviada += 'Host: localhost\r\n'
messagem_enviada += '\r\n'
```

Figura 8 – Requisição utilizando versão 2.0 do HTTP

Mensagem recebida pelo cliente (enviada pelo servidor), contendo um arquivo html informando que esta versão do HTTP não é suportada pelo servidor:

```
HTTP/1.1 505 HTTP Version Not Supported
Date: Wed Aug 18 16:23:45 2021
Server: YD-Server Win 10
Content-Type: text/html

                        <!DOCTYPE html>
                        <html lang="en">
                        <head>
                            <meta charset="UTF-8">
                            <title>505 HTTP Version Not Supported</title>
                        </head>
                        <body>
                            <h1>505 HTTP Version Not Supported</h1>
                            <h2>Versão do HTTP utilizada não é suportada neste servidor<h2>
                        </body>
                        </html>
```

Figura 9 – Erro 505 HTTP Version Not Supported

4.3.4 200 OK

Para o desenvolvimento das respostas a requisições válidas, foram divididas em três tipos:

4.3.4.1 Requisição de Pastas

Caso o cliente solicite uma pasta que contenha um arquivo chamado index.html ou index.htm, ele deve mostrar esse arquivo.

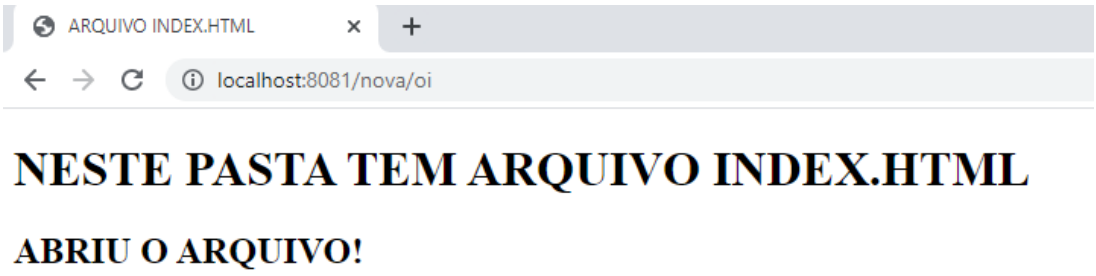

Figura 10 – Pasta com arquivo Index.html

4.3.4.2 Requisição de Diretório de Pastas

Caso o cliente solicite uma pasta que não contenha um arquivo chamado index.html ou index.htm, ele deve mostrar uma lista dos arquivos e pastas contidos, com tamanho e data da última modificação, na qual ele poderá navegar.

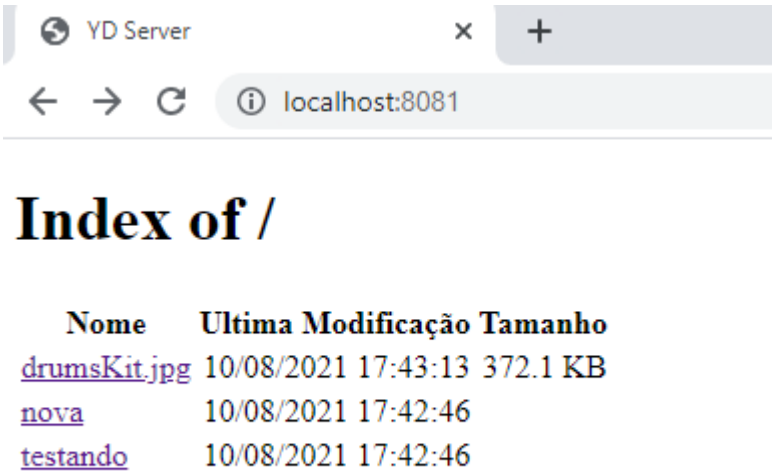

Figura 11 – Diretório das Pastas

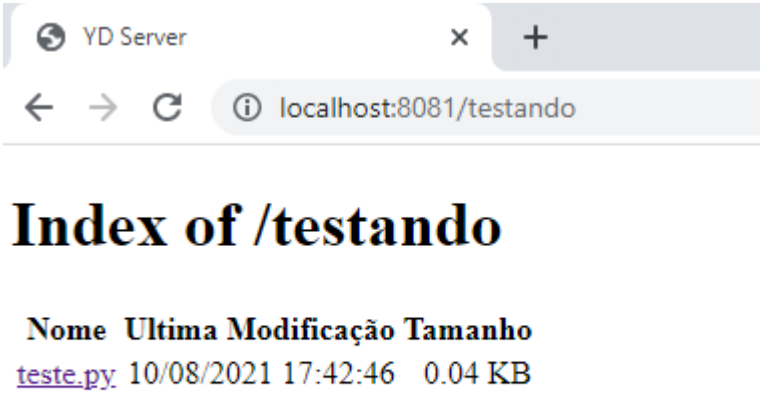

Figura 12 – Navegando nas pastas

4.3.4.3 Requisição de arquivos

Caso o cliente solicite um arquivo específico, este será apresentado na tela. Se o arquivo for grande, é feito o download do mesmo.

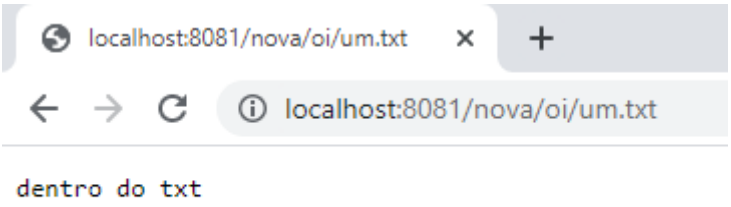

Figura 13 – Arquivo de texto

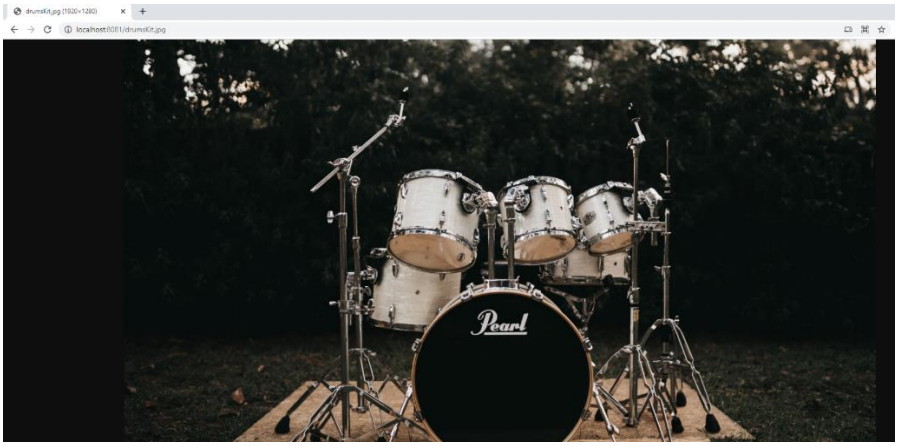

Figura 14 – Arquivo de imagem

5. CONCLUSÃO

Os projetos foram importantes para mostrar na prática como os protocolos da camada de aplicação e de transporte funcionam. Entender o papel e a importância de cada protocolo, ajuda na compreensão de como as redes funcionam como um todo.

Para que as aplicações da internet funcionem corretamente, milhares de servidores foram estruturados e desenvolvidos. A requisição de dados e envio de pacotes fazem parte do cotidiano das pessoas, porém, não são percebidos facilmente. A criação de um servidor, permite uma melhor visualização do como as mensagens são trocadas.

6. REFERÊNCIAS

KUROSE, James F.; ROSS, Keith W. Redes de computadores e a Internet: uma abordagem top-down. 6.ed. São Paulo (SP): Pearson Education do Brasil, 2013.