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

GUSTAVO PRAZERES PAZ DO NASCIMENTO - gppn@cin.ufpe.br
VINICIUS SANTIAGO BEZERRA - vsb@cin.ufpe.br
YURI RODRIGUES DE ALENCAR LOPES - yral@cin.ufpe.br

PROJETO DE DESENVOLVIMENTO DE SERVIDORES (TCP/UDP) COM SOCKETS EM PYTHON

RECIFE, PE
2020

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

GUSTAVO PRAZERES PAZ DO NASCIMENTO - gppn@cin.ufpe.br
VINICIUS SANTIAGO BEZERRA - vsb@cin.ufpe.br
YURI RODRIGUES DE ALENCAR LOPES - yral@cin.ufpe.br

PROJETO DE DESENVOLVIMENTO DE SERVIDORES (TCP/UDP) COM SOCKETS EM PYTHON

Relatório do desenvolvimento de servidores TCP e UDP com Sockets em python, 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
2020

SUMÁRIO

1. INTRODUÇÃO

Se dois computadores são capazes de compartilhar informações entre si ou até mesmo compartilhar seus recursos de hardware, não importando o meio, dizemos que temos uma rede de computadores. Nesse conceito, também pode estar englobado outros dois, que é o de cliente e servidor.

Os servidores ajudam a centralizar e organizar a infraestrutura de computadores, impressoras e qualquer tipo de equipamento que se conectará à rede interna da organização. Consequentemente, estamos falando de acesso, manutenção, gerenciamento e segurança.

No que se diz respeito a internet, essa se baseia, no geral, em requisições e respostas. Por trás desse processo, existem diversas camadas, mas uma fundamental é a camada de transporte, que é responsável pela transferência de dados entre diferentes máquinas e possui dois protocolos fundamentais: o TCP e o UDP (Alura, 2019).

Ainda segundo Alura (2019), O protocolo TCP é, talvez, o mais utilizado na camada de transporte para aplicações na Web. Diferente do UDP, o TCP é voltado à conexão e tem como garantia a integridade e ordem de todos os dados.

Uma vez que todos esses conceitos foram apresentados na disciplina de Redes de Computadores, esse relatório tem o objetivo de trazer resultados da aplicação prática do conteúdo, com o desenvolvimento de dois servidores com sockets (UDP e TCP), utilizando a linguagem python.

2. CRONOGRAMA

O cronograma das atividades está listado na Tabela 1.

Atividade	**Data**
Divulgação dos Projetos via Google Classroom	22/09/2020
Início do Projeto 1	24/09/2020
Conclusão do Projeto 1	30/10/2020
Início do Projeto 2	02/11/2020
Início da realização do relatório	02/11/2020
Conclusão do Projeto 2	15/11/2020
Conclusão do relatório	17/11/2020
Entrega e apresentação dos projetos	19/11/2020

Tabela 1 - Cronograma de atividades

3. PROJETO 1 - QUIZ COMPETITIVO - UDP

3.1 DESCRIÇÃO E SOLUÇÃO DOS PROBLEMAS

O projeto 1 trata do desenvolvimento online de jogos e respostas, sendo solicitado o desenvolvimento de um protocolo da camada de aplicação, que será especificado neste relatório, que funcione no protocolo de transporte UDP, pois este é voltado a conexões e é capaz de gerenciar a competição.

Iniciamos realizando um fluxograma com o funcionamento do quiz para auxiliar no entendimento do protocolo e implementar, como mostra o Anexo 1. O diagrama traz o papel tanto do cliente, quanto do servidor na competição.

Uma vez que foram definidos os papéis de cada parte que compõe o jogo, iniciou-se a leitura da documentação de *sockets* para python e em seguida a implementação.

3.2 MÉTODOS

O primeiro método utilizado foi a definição do nosso protocolo da camada de aplicação. Criamos números para trocar informações entre o servidor e o cliente. Os códigos e as descrições estão na Tabela 2.

Código	**Descrição**
400	Resposta incorreta
500	Resposta correta
800	Tempo esgotado
900	Outro jogador acertou

Tabela 2 - Códigos do protocolo da aplicação - Projeto 1

Após essas definições, partimos para implementação do código, colocando o socket do servidor UDP em modo passivo para o servidor poder ficar “escutando” a porta.

Na implementação também foram utilizadas várias estruturas de dados: Listas, para armazenar as tuplas com IP e Porta, além da pontuação dos jogadores; Tuplas, para guardar o IP e a porta de cada jogador, e Arquivos, no formato .txt para agrupar as perguntas e respostas a serem lidas quando necessário.

Por fim, foram utilizadas diversas bibliotecas, como: Threading, que permite que o servidor controle o jogo, o que foi fundamental para que conseguíssemos tornar nossa aplicação assíncrona, ou seja, para evitar que enquanto o servidor atende uma conexão ele fica dedicado a ela e possa tratar da

nova conexão que está chegando; *Time,* para atrasar o código em alguns pontos, *e Sys,* para encerrar o programa.

3.3. TESTES

A aplicação foi testada com até 5 jogadores, conectados na mesma rede via VPN do CIN (PPTP). Para execução do código, utilizamos o VirtualBox para hospedar uma máquina virtual como servidor e nossos computadores como clientes. Os testes apresentaram resultados satisfatórios para o jogo e atendeu todas as exigências do projeto.

3.4 EXECUÇÃO DA PARTIDA

Para executar a partida, primeiro é necessário configurar os IP's, tanto do servidor quanto dos clientes. O número de jogadores está fixo como cinco, mas pode ser alterado no código.

Uma vez iniciado o servidor, ele começa a aguardar as requisições dos clientes e só começa o jogo quando atingir 5 jogadores. A Tabela 3 mostra o comportamento do terminal do cliente e o do servidor em diversas etapas do *game*.

Processo *Descrição do processo*	**Terminal do Cliente**	**Terminal do Servidor**

Tela de início/cadastro
 	
Telas exibidas no início do programa
	`BEM VINDO AO JOGUINHO`
	`Teste sua habilidade em geografia acertando o maior número de capitais!`
	
	`Para iniciar, digite seu nome!`
	`Nome:`
		`Servidor UDP escutando requisições...`

Perguntas
 	
Servidor envia comando 'start' para cliente, o jogo inicia e começa a troca de mensagens de perguntas e respostas
	`Iniciando competição...`
	
	`Partida n° 1`
	`Pergunta: Capital do Canadá?`
	`Insira sua resposta:`
		`Iniciando competição...`
		
		`Partida n°: 1`

Acertando resposta
 	
Caso a capital digitada seja correta, o cliente que acertou
	`Partida n° 2`
	
	`Pergunta: Capital da Venezuela?`
		`Partida n°: 2`
		
		`Recebeu caracas de 192.168.1.65`

ganha 25 pontos	Insira sua resposta: caracas Resposta correta! Próxima pergunta -->	192.168.1.65 acertou
Errando resposta Caso seja a capital errada, o cliente perde 5 pontos e recebe uma nova chance para tentar acertar Ele pode tentar até acertar, outro jogador acertar ou o tempo esgotar.	Partida n° 3 Pergunta: Capital de Israel? Insira sua resposta: recife Resposta incorreta.. tente novamente: olinda Resposta incorreta.. tente novamente: Tempo esgotado. Próxima pergunta -->	Partida n°: 3 Recebeu recife de 192.168.1.65 Recebeu olinda de 192.168.1.65 Tempo esgotado
Outro cliente acertou Se o jogador não acertar ou não enviar uma resposta e outro jogador acertar, esse outro jogador ganha 25 pontos e o anterior não pode mais enviar tentativas	Partida n° 4 Pergunta: Capital da Alemanha? Insira sua resposta: Outro jogador acertou. Próxima pergunta -->	Partida n°: 4 Recebeu berlin de 192.168.1.32 192.168.1.32 acertou
Tempo esgotado Caso se passe 10 segundos e nenhum jogador acerte ou não envie respostas, o servidor encerra a rodada e todos os jogadores perdem 1 ponto	Partida n° 5 Pergunta: Capital de Portugal? Insira sua resposta: Tempo esgotado	Partida n° 5 Tempo esgotado

Fim da partida e exibição do ranking Após o fim do jogo, um ranking com as pontuações é exibido. O programa do cliente encerra e servidor reinicia automaticamente	Fim de jogo! Pontuações da partida: Gustavo = 37 Yuri = 113 Vinicius = 75 Obrigado por jogar. Volte sempre!	Fim de jogo! Pontuações da partida: Gustavo = 37 Yuri = 113 Vinicius = 75 Servidor UDP escutando requisições...
Tentativa de cadastro quando partida já começou Se o jogo já começou, novos clientes não devem se cadastrar até o término do atual e início de um novo	BEM VINDO AO JOGUINHO Teste sua habilidade em geografia acertando o maior número de capitais! Para iniciar, digite seu nome! Nome:Gustavo Partida Já Iniciou. Tente novamente em instantes! Pressione qualquer tecla para finalizar	Partida n° x Alguém não cadastrado tentou se conectar

Tabela 3 - Terminal do cliente e servidor em diversas etapas do jogo.

4. PROJETO 2 - SERVIDOR WEB - TCP

4.1 DESCRIÇÃO E SOLUÇÃO DOS PROBLEMAS

O projeto 2 trata do desenvolvimento de um servidor web, implementando o protocolo padronizado HTTP/1.1, apenas com o método GET. O protocolo de transporte foi pré-definido como sendo o TCP, pois facilita e possibilita a comunicação e é Responsável por receber os dados camada anterior, verificar a integridade deles, organizá-los e dividi-los em pacotes menores, que serão enviados ao cliente.

4.2 MÉTODOS

Mais uma vez, iniciamos definindo nosso protocolo da camada de aplicação. Os códigos e as descrições do protocolo HTTP/1.1 estão na Tabela 3. Com exceção do código 200, o servidor envia um arquivo html personalizado informando o respectivo erro.

Código	**Descrição**
200 OK	Requisição bem-sucedida, objeto requisitado será enviado
400 Bad Request	Mensagem de requisição não entendida pelo servidor, nesse caso o cliente escreveu a mensagem de requisição com algum erro de sintaxe
404 Not Found	Documento requisitado não localizado no servidor
501 Not implemented	Caso a requisição seja diferente de GET
505 HTTP Version Not Supported	Versão do HTTP utilizada não é suportada neste servidor

Tabela 3 - Códigos do protocolo da aplicação - Projeto 2

Em seguida, iniciamos a implementação do servidor com *sockets TCP*, importando a biblioteca, definindo IP e Porta e criando funções para ele ficar “escutando” as requisições feitas pelo cliente.

Uma vez que o servidor está escutando, ele pode realizar o *three-way handshake* com o cliente, ou seja, primeiro o cliente verifica a conexão com o servidor (SYN), caso a resposta seja positiva (SYN-ACK), ele envia a requisição e então o servidor verifica o método, se é o GET ou não, ele verifica o caminho do arquivo que o cliente quer e a versão do HTTP, como mostra o exemplo da Figura 1.

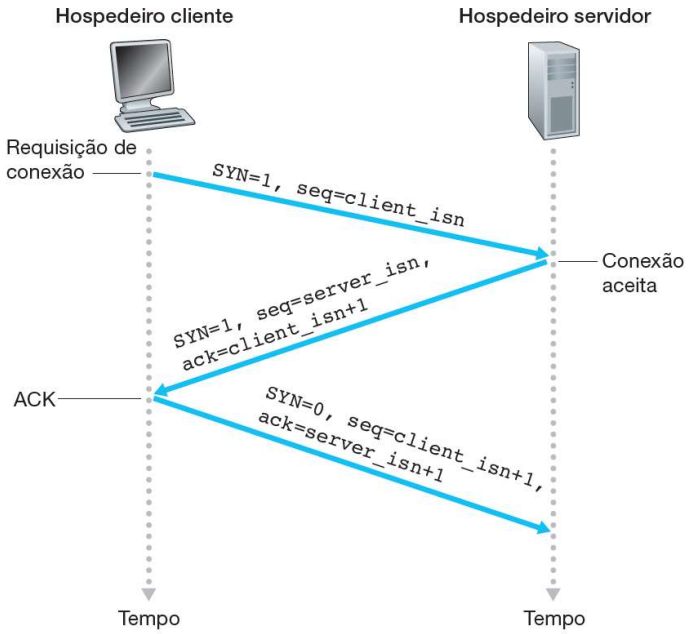

Figura 1 - Exemplo de *three-way handshake*

Caso a requisição seja GET, o servidor verifica o caminho do arquivo/pasta e retorna para o browser. Caso contrário, retorna o erro 501, pois só requisições desse tipo são permitidas. A versão do HTTP também é verificada, se não for 1.1 ou 1.0, retorna erro 505

Nos casos em que o 200 é retornado, o servidor manda mensagem do status, a versão, a data de acesso, o nome do servidor, content-type, content lance, por fim o arquivo.

Para diretório de pastas, uma lista é criada, com todos os arquivos presente no diretório analisado, que é adicionada ao código HTML a ser enviado como resposta, semelhante a como ocorre no Apache.

Os principais métodos utilizados no servidor para garantir o funcionamento incluem a biblioteca *mimetypes* para obter a extensão de um arquivo e obter o Content-Type a ser posteriormente enviado na resposta da requisição HTTP, a função *os.path()* da biblioteca *OS* para obtermos os diretórios atuais e dos arquivos presentes nos diretórios, a biblioteca time, além de listas e tuplas.

4.3 TESTES

Para os testes, utilizamos a VPN do Centro de Informática para simularmos uma rede local para que o servidor se comunique com seus clientes. Também utilizamos o Hamachi para esse propósito em momentos de instabilidade da VPN do CIn . Em todos os casos, definimos os respectivos IP's e a porta fixa 8080.

4.4 EXECUÇÃO - CLIENTE

Serão listados em tópicos a resposta recebida no browser pelo servidor quando enviada diversas solicitações que retornam com sucesso - código 200 OK.

4.4.1. Diretório de pastas

Ao digitar o caminho de um diretório, será retornado uma lista de arquivos e pastas contidos nele, o qual você pode navegar, como mostra a Figura 2. Essa ação simula o que acontece em um servidor apache. Se for digitado apenas o IP/Porta, o servidor retorna o diretório raiz.

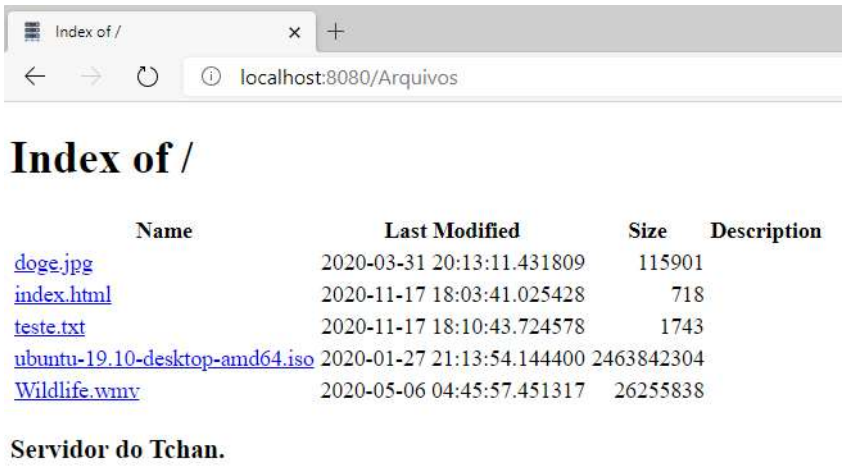

Figura 2 - Lista de arquivos/pastas contidos no diretório

4.4.2. Requisição de arquivo de texto e HTML com Imagem

Na Figura 3, podemos ver que ao requerer um arquivo HTML, o servidor enviará para o navegador, bem como as imagens contidas nele. Já a Figura 4 traz uma requisição de arquivo de texto.

Figura 3 - Requisição de HTML com Imagem

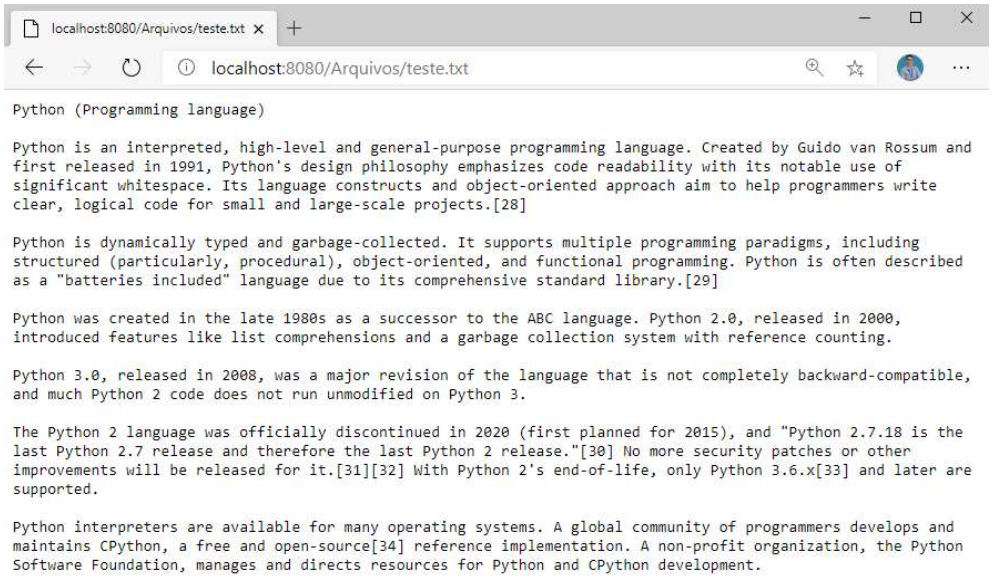

Figura 4 - Requisição de arquivo de texto

4.4.3. Outros arquivos

Para requisições de arquivos pesados, como vídeos (250Mb) ou outros, como por exemplo, a ISO do ubuntu 19.10(2.37GB), o navegador realiza o download, como ilustrado na Figuras 5 e 6.

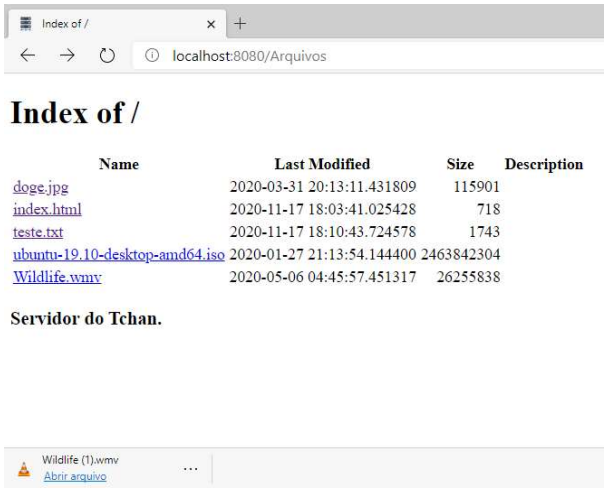

Figura 5 - Requisição de arquivo de vídeo com 250MB.

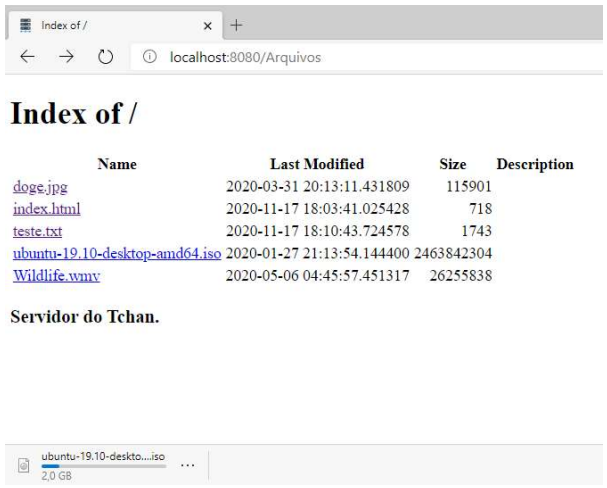

Figura 6 - Requisição da ISO do ubuntu 19.10(2.37GB).

4.5 ERROS 501/505

Para os casos em que as requisições não são do tipo GET ou a versão do HTTP não é 1.0 ou 1.1, foram implementados os erros 501 e 505, respectivamente.

A fim de verificar se o servidor estava corretamente fazendo essas validações, foram criado códigos que realizam requisições da mesma maneira que um navegador faria, só que com a possibilidade de personalizar qual será o tipo da requisição e a versão do HTML com maior facilidade, como vemos a seguir:

```
from socket import socket, AF_INET, SOCK_STREAM
sock = socket(AF_INET, SOCK_STREAM)

address = 'localhost',8080
sock.connect(address)

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

```
sock.send(message.encode())

data_recieved = sock.recv(2000)
print(data_recieved.decode())
```

A resposta do servidor para as requisições diferentes de GET e versões de HTTP não compatíveis, pode ser vista na Figura 7 e 8, respectivamente..

```
HTTP/1.1 501 not implemented
Date: Wen 21 Oct 2020 12:10:30 GMT
Allow: GETServer: servidordotchan/0.0.1 (Windows)
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>
                    </body>
                    </html>
```

Figura 7 - Método diferente de GET.

```
HTTP/1.1 505 HTTP Version Not Supported
Date: Wen 21 Oct 2020 12:10:30 GMT
Server: servidordotchan/0.0.1 (Windows)
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>
                    </body>
                    </html>
```

Figura 8 - Erro html version not supported, caso a versão solicitada do HTTP não seja HTTP/1.0 ou HTTP/1.1

5. CONCLUSÃO

Como foi mostrado neste trabalho, para que parte das aplicações funcionem, os servidores são fundamentais, desempenhando papéis como execução de programas de forma centralizada, além de armazenar e compartilhar arquivos, através de uma rede local ou remota.

Neste projeto, foram desenvolvidos 2 tipos de servidores, funcionando com diferentes protocolos da camada de transporte, e isso contribuiu com a melhora do nosso conhecimento, pois tivemos contato com as mais diversas soluções práticas aplicadas, antes só vistas de forma teórica.

REFERÊNCIAS

[1] Alura. **Quais as diferenças entre UDP e TCP?.** Disponível em <https://www.alura.com.br/artigos/quais-as-diferencas-entre-o-tcp-e-o-udp.>. Acessado em 11 de outubro de 2020.

[2] James F. Kurose, Keith W. Ross. **Redes de computadores e a Internet: uma abordagem top-down**. 6. ed. – São Paulo: Pearson Education do Brasil, 2013.

[3] Python 3.9.0 documentation. **socket - Low-level networking interface.** Disponível em <https://docs.python.org/3/library/socket.html> . Acessado em 20 de outubro de 2020.

[4] Python 3.9.0 documentation. **threading — Thread-based parallelism.** Disponível em <https://docs.python.org/3/library/threading.html> . Acessado em 20 de outubro de 2020.

[5] Python 3.9.0 documentation. mimetypes — **Map filenames to MIME types.** Disponível em <https://docs.python.org/3/library/mimetypes.html>. Acessado em 10 de novembro de 2020.

ANEXO A