RELATÓRIO

PI V - 5º Semestre de Bacharelado em Ciência da Computação

Humberto Vieira de Castro
Mario Roberto Suruagy de Castro

Etapa 1

Para o projeto da disciplina de Projeto Integrador V do curso de Bacharelado em Ciência da Computação foi proposto o desenvolvimento de uma simulação de um sistema de defesa anti-aérea.

A primeira etapa do projeto consistiu em implementar uma sistema de comunicação Cliente-Servidor entre dois computadores utilizando uma biblioteca de socket/winsock e o protocolo TCP/IP.

Para o desenvolvimento das aplicações foi escolhido a linguagem de programação Python 3.4 por se tratar de uma linguagem que os integrantes do grupo não haviam trabalhado anteriormente e tinham interesse de se familiarizar. Para a comunicação de socket/winsock foi utilizada as bibliotecas Websockets 3.0 e Websocket-Client 0.35.0.

O sistema operacional utilizado para o desenvolvimento foi o OS X El Captain. O controle de versionamento do código foi realizado através do Git.

Tecnologias

O TCP/IP (Transmission Control Protocol) é o principal protocolo de comunicação entre computadores utilizado na internet. Este protocolo, na verdade é um conjunto de protocolos divididos em quatro camadas aplicação, transporte, rede e interface.

O websocket é uma tecnologia que permite a comunicação bidirecional sobre um único socket TCP/IP. Baseado no TCP/IP, ela foi projetada para browsers e servidores web que suportam o HTML 5, entretanto, pode ser usado por qualquer cliente ou servidor.

Instalação

Como esclarecido anteriormente, o projeto foi desenvolvido no sistema operacional OS X El Captain, porém, o processo de instalação pode ser facilmente adaptado para plataformas Linux.

Primeiramente foi necessário instalar o Python 3.4 na maquina do desenvolvedores. No OS X foi utilizado o gerenciador de pacotes *homebrew* para este fim:

```
brew install python3
```

Em seguida foram instaladas as bibliotecas utilizadas:

```
sudo pip3 install websockets
sudo pip3 install websocket-client
```

Para executar o código utiliza-se o comando:

```
python3 nomedoarquivo.py
```

Desenvolvimento

Primeiramente foi realizado por parte dos integrantes do grupo um estudo da linguagem Python, uma vez que nenhum integrante havia trabalhado com ela anteriormente. O objetivo do estudo foi se familiarizar com a sintaxe da linguagem para possibilitar o restante do desenvolvimento do sistema.

Em seguida, o grupo iniciou uma pesquisa de bibliotecas de socket para utilizar no projeto e tutoriais de utilização destas. Encontramos um tutorial para o desenvolvimento de um sistema

de chat com a biblioteca *websockets* e *websockets-client*. O tutorial foi primeiramente implementado com o objetivo de entender o seu funcionamento, e em seguida adaptado para os fins do projeto.

Foram desenvolvidos dois códigos distintos. Um para a aplicação de servidor *server.py* e outro para a aplicação de cliente *client.py*. A aplicação *server.py* cria o servidor e espera pela conexão de uma aplicação *client.py*. Assim que a conexão é realizada, inserindo o IP do servidor no código do cliente, o cliente passa a aguardar mensagens que serão enviadas pelo servidor. O servidor por sua vez possibilita a um usuário o input de um texto a ser enviado e o envia para o cliente, que manda uma resposta de sucesso ao receber a mensagem. O servidor calcula o tempo de envio da mensagem através da diferença entre o horário de envio e o horário de recebimento da resposta de sucesso, e é capaz de armazenar os tempos de todas as mensagens enviadas para apresentar uma media de tempo de envio.

Caso ocorra algum erro no envio e o cliente não receba a mensagem corretamente, um código de erro é enviado na resposta para o servidor e a mensagem é enviada novamente. Se o tempo de espera para a resposta superar um valor pré determinado o envio é cancelado com um erro de tempo excedido.

Resultados.

Saida do *server.py:*

```
Servidor iniciado!
Esperando conexão de cliente...
Radar conectado.

Digite sua mensagem: Humberto Vieira

Enviando....
Enviado! (tempo total: 0.010287046432495117)
tempo medio: 0.010287046432495117

Digite sua mensagem: Humberto Vieira

Enviando....
Enviado! (tempo total: 0.07540607452392578)
tempo medio: 0.04284656047821045

Digite sua mensagem: Humberto Vieira

Enviando....
Enviado! (tempo total: 0.006022214889526367)
tempo medio: 0.030571778615315754

Digite sua mensagem: Humberto Vieira

Enviando....
Enviado! (tempo total: 0.04142308235168457)
tempo medio: 0.03328460454940796

Digite sua mensagem: Humberto Vieira

Enviando....
Enviado! (tempo total: 0.2154388427734375)
tempo medio: 0.06971545219421386

Digite sua mensagem: Humberto Vieira

Enviando....
Enviado! (tempo total: 0.17885684967041016)
```

```
tempo medio: 0.08790568510691325
```

Saida do client.py:

```
Iniciando...
Radar Conectado
Aguardando mensagem...

Mensagem: Humberto Vieira

Aguardando mensagem...

Mensagem: Humberto Vieira

Aguardando mensagem...

Mensagem: Humberto Vieira

Aguardando mensagem...

Mensagem: Humberto Vieira

Aguardando mensagem...

Mensagem: Humberto Vieira

Aguardando mensagem...

Mensagem: Humberto Vieira

Aguardando mensagem...
```

Referencias.

git do projeto:
https://github.com/afrodev/BCC-1S16-PI5-MISSEIS

websockets:
https://pypi.python.org/pypi/websockets

websocket-client:
https://pypi.python.org/pypi/websocket-client/

async.io
https://docs.python.org/3/library/asyncio.html

Tutorial utilizado:
http://junglecoders.blogspot.com.br/2014/08/servidor-de-chat-com-websockets-e.html

Anexo 01 - server.py

```
# importando todas as bibliotecas necessárias
import asyncio # Para métodos que rodam assincronamente
import websockets # Cuida dos métodos de websocket
import time
# import shlex

# Classe Servidor - Cuida das funções do servidor
class Servidor:
    def __init__(self):
        self.tempos = []

    # Colocando esse codigo antes, executamos a função em modo assincrono
    @asyncio.coroutine
    def conecta(self, websocket, path):
        cliente = Cliente(self, websocket, path)

        print("Radar conectado.\n")
        yield from cliente.gerencia() # Com o yield podemos garantir que será executado
de forma sequencial, colocando em uma fila.

    # Função para desconectar o cliente que entrou no servidor.
    def desconecta(self, cliente):
        print("Radar desconectado.")

# Classe Cliente - Cuida das funçoes de transferencia de mensagem
class Cliente:
    # Inicializa a classe do websocket
    def __init__(self, servidor, websocket, path):
        self.cliente = websocket
        self.servidor = servidor

    # De forma assincrona, gerencia as conexões dos usuários no websocket
    @asyncio.coroutine
    def gerencia(self):
        try:
            # De forma sequencial, envia a todos os usuários a mensagem
            yield from self.envia("Radar Conectado")

            while True: # Enquanto a mensagem não chegar o servidor fica esperando
                # Escreve a mensagem vinda do teclado
                mensagem = input("Digite sua mensagem: ")

                #Coloca o tempo na hora de enviar a mensagem e envia
                tempoEnvio = time.time()
                tempoTotal = 0;
                intervaloLimite = 2;

                mensagemTotal = mensagem
                result = "1";

                print("\nEnviando....")

                while not result == "0" and tempoTotal < intervaloLimite:
                    yield from self.envia(mensagemTotal)

                    # Recebe o retorno e pega o tempo que demora para ele retornar
                    result = yield from self.recebe()

                    tempoRecebimento = time.time()
                    tempoTotal = tempoRecebimento - tempoEnvio

                if result != "0":
                    print("Falha no envio, tempo limite excedido.")
```

```
                else:
                    servidor.tempos.append(tempoTotal);
                    print("Enviado! (tempo total: {0})".format(tempoTotal))
                    total = 0
                    for tempo in servidor.tempos:
                        total += tempo

                    print("tempo medio: {0}".format(total / len(servidor.tempos)) + "\n")

        except Exception:
            print("Erro")
            raise
        finally:
            self.servidor.desconecta(self)

    # Envia mensagem pelo websocket
    @asyncio.coroutine
    def envia(self, mensagem):
        yield from self.cliente.send(mensagem)

    # Recebe mensagem via websocket
    @asyncio.coroutine
    def recebe(self):
        mensagem = yield from self.cliente.recv()
        return mensagem


# Criando o servidor efetivamente
servidor = Servidor()
loop = asyncio.get_event_loop()

# Inicializa o servidor com web socket
start_server = websockets.serve(servidor.conecta, '0.0.0.0', 8766) # Usa-se 0.0.0.0 para
pegar o ip do computador local

# Usa um trycatch para deixar rodando infinitamente o servidor de websocket
try:
    print("Servidor iniciado!")
    print("Esperando conexão de cliente...")
    loop.run_until_complete(start_server)
    loop.run_forever()
finally:
    start_server.close()
```

Anexo 02 - client.py

```
# importa o essencial
from websocket import create_connection
import time

#Inicializa a conexão com o WebSocket
ws = create_connection("ws://172.20.10.3:8766")

# Cuida do retorno da conexão de abertura com o websocket
print("Iniciando...")
result = ws.recv()
print(result)

# Envia mensagem de conexao no servidor
print("Aguardando mensagem...")
# Crio um laço infinito para enviar mensagens que envia mensagens para o servidor
while True:
    mensagem = ws.recv()
    if mensagem:
        print("\nMensagem: {0}".format(mensagem) + "\n")
        print("Aguardando mensagem...")
        ws.send("0")
    else:
        ws.send("1")

ws.close()

ws.run_forever()
```