Guilherme Morbeck Rodrigues

TP – Redes de Computadores

Echo Client and Server

Os códigos echo-server.py e echo-client.py criam uma comunicação cliente e servidor implementando a biblioteca socket da linguagem Python. O servidor instancia um objeto socket e o coloca para aguardar por uma conexão. O cliente também instancia seu socket e faz conexão com algum servidor, assim, podendo transferir dados, no caso, uma string. Ao estabelecer a conexão, o servidor recebe os dados, envia uma resposta ao cliente e termina a conexão.

```
gguimbck@guilherme-ubuntu:~/Documentos/Redes de Computadores/TP$ ./echo-server.p
y
Guilherme Morbeck Rodrigues - TP Redes de Computadores
Connected by:  ('127.0.0.1', 58356)
gguimbck@guilherme-ubuntu:~/Documentos/Redes de Computadores/TP$
```

```
gguimbck@guilherme-ubuntu:~/Documentos/Redes de Computadores/TP$ ./echo-client.p
y
Guilherme Morbeck Rodrigues - TP Redes de Computadores
Received b'Hello,world'
gguimbck@guilherme-ubuntu:~/Documentos/Redes de Computadores/TP$
```

Os códigos foram executados juntos e é possível verificar a conexão pelo IP e pela porta usada no terminal que executa o servidor. No terminal do cliente, é possível verificar a resposta do servidor, que foi a própria string.

```
gguimbck@guilherme-ubuntu:~$ lsof -i -n
COMMAND    PID     USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
firefox   2956 gguimbck   38u  IPv4 140932      0t0  TCP 192.168.15.16:52894->34.117.237.239:https (ESTABLISHED)
firefox   2956 gguimbck   40u  IPv6 138203      0t0  TCP [2804:7f2:2a8b:b56f:a92c:1b7e:13ae:e63d]:38096->[2800:3f0:4001:81e::200e]:https (ESTABLISHED)
firefox   2956 gguimbck   49u  IPv6 139629      0t0  TCP [2804:7f2:2a8b:b56f:a92c:1b7e:13ae:e63d]:56486->[2a03:2880:f233:c5:face:b00c:0:167]:https (ES
TABLISHED)
firefox   2956 gguimbck   76u  IPv4 143590      0t0  TCP 192.168.15.16:48998->52.38.70.232:https (ESTABLISHED)
firefox   2956 gguimbck  126u  IPv6 134085      0t0  TCP [2804:7f2:2a8b:b56f:a92c:1b7e:13ae:e63d]:51498->[2620:1ec:27::cafe:1185]:https (ESTABLISHED)
firefox   2956 gguimbck  127u  IPv6 139700      0t0  TCP [2804:7f2:2a8b:b56f:a92c:1b7e:13ae:e63d]:55246->[2800:3f0:4001:823::2004]:https (ESTABLISHED)
firefox   2956 gguimbck  155u  IPv6 140596      0t0  TCP [2804:7f2:2a8b:b56f:a92c:1b7e:13ae:e63d]:40526->[2800:3f0:4001:817::200e]:https (ESTABLISHED)
firefox   2956 gguimbck  156u  IPv6 128806      0t0  TCP [2804:7f2:2a8b:b56f:a92c:1b7e:13ae:e63d]:56442->[2a03:2880:f233:c5:face:b00c:0:167]:https (ES
TABLISHED)
firefox   2956 gguimbck  159u  IPv4  46631      0t0  TCP 192.168.15.10:36882->52.41.2.143:https (ESTABLISHED)
firefox   2956 gguimbck  161u  IPv6 138912      0t0  TCP [2804:7f2:2a8b:b56f:a92c:1b7e:13ae:e63d]:55250->[2800:3f0:4001:823::2004]:https (ESTABLISHED)
firefox   2956 gguimbck  287u  IPv6 138969      0t0  TCP [2804:7f2:2a8b:b56f:a92c:1b7e:13ae:e63d]:52280->[2800:3f0:4001:80f::2003]:https (ESTABLISHED)
firefox   2956 gguimbck  290u  IPv6 138166      0t0  TCP [2804:7f2:2a8b:b56f:a92c:1b7e:13ae:e63d]:60972->[2800:3f0:4001:812::2003]:https (ESTABLISHED)
firefox   2956 gguimbck  292u  IPv6 138178      0t0  TCP [2804:7f2:2a8b:b56f:a92c:1b7e:13ae:e63d]:40442->[2800:3f0:4001:808::200e]:https (ESTABLISHED)
firefox   2956 gguimbck  299u  IPv6 140113      0t0  TCP [2804:7f2:2a8b:b56f:a92c:1b7e:13ae:e63d]:46924->[2800:3f0:4001:820::2002]:https (ESTABLISHED)
firefox   2956 gguimbck  304u  IPv6 138180      0t0  TCP [2804:7f2:2a8b:b56f:a92c:1b7e:13ae:e63d]:50912->[2800:3f0:4001:818::2002]:https (ESTABLISHED)
firefox   2956 gguimbck  305u  IPv6 138997      0t0  TCP [2804:7f2:2a8b:b56f:a92c:1b7e:13ae:e63d]:37104->[2800:3f0:4001:813::2002]:https (ESTABLISHED)
firefox   2956 gguimbck  308u  IPv6 139005      0t0  TCP [2804:7f2:2a8b:b56f:a92c:1b7e:13ae:e63d]:33808->[2800:3f0:4001:812::200e]:https (ESTABLISHED)
firefox   2956 gguimbck  312u  IPv4 138190      0t0  TCP 192.168.15.16:50216->142.250.219.2:https (ESTABLISHED)
firefox   2956 gguimbck  320u  IPv6 140589      0t0  TCP [2804:7f2:2a8b:b56f:a92c:1b7e:13ae:e63d]:48282->[2800:3f0:4001:81e::2001]:https (ESTABLISHED)
firefox   2956 gguimbck  321u  IPv6 140590      0t0  TCP [2804:7f2:2a8b:b56f:a92c:1b7e:13ae:e63d]:54928->[2800:3f0:4001:81d::2003]:https (ESTABLISHED)
plugin_ho 3625 gguimbck   19u  IPv4  56998      0t0  TCP 192.168.15.10:45308->45.55.41.223:http (CLOSE_WAIT)
python3   6697 gguimbck    3u  IPv4 143720      0t0  TCP 127.0.0.1:65432 (LISTEN)
gguimbck@guilherme-ubuntu:~$
```

Com o comando lsof, foi possível verificar o socket aguardando uma conexão, com o IP 127.0.0.1 e na porta 65432, que foram definidos no código. Também é possível verificar o padrão IPv4 e o uso de TCPs, que foram definidos no código.

Multi-Connection Client and Server

Os códigos multiconn-server.py e multiconn-client.py foram implementados com o objetivo de lidar com uma múltipla conexão entre cliente e servidor.

```
gguimbck@guilherme-ubuntu:~/Documentos/Redes de Computadores/TP$ ./multiconn-ser
ver.py 127.0.0.1 65432
Guilherme Morbeck Rodrigues - TP Redes de Computadores
listening on ('127.0.0.1', 65432)
```

Assim como na primeira implementação, o servidor é criado e aguardar por alguma conexão.

```
gguimbck@guilherme-ubuntu:~/Documentos/Redes de Computadores/TP$ ./multiconn-cli
ent.py 127.0.0.1 65432 2
Guilherme Morbeck Rodrigues - TP Redes de Computadores
starting connection 1 to ('127.0.0.1', 65432)
starting connection 2 to ('127.0.0.1', 65432)
sending b'Message 1 from client.' to connection 1
sending b'Message 1 from client.' to connection 2
sending b'Message 2 from client.' to connection 1
sending b'Message 2 from client.' to connection 2
received b'Message 1 from client.Message 2 from client.' from connection 1
closing connection 1
received b'Message 1 from client.Message 2 from client.' from connection 2
closing connection 2
gguimbck@guilherme-ubuntu:~/Documentos/Redes de Computadores/TP$
```

Ao executar o código do cliente, o socket é criado que inicia conexão com o servidor, mas diferentemente da outra implementação, mais de um socket é criado e mais de uma conexão é feita.

Cada conexão envia uma mensagem. No print da tela é possível verificar que duas conexões foram feitas com o servidor, que mandou uma resposta ao cliente para cada conexão e as conexões foram terminadas.

```
gguimbck@guilherme-ubuntu:~/Documentos/Redes de Computadores/TP$ ./multiconn-ser
ver.py 127.0.0.1 65432
Guilherme Morbeck Rodrigues - TP Redes de Computadores
listening on ('127.0.0.1', 65432)
accepted connection from ('127.0.0.1', 59144)
accepted connection from ('127.0.0.1', 59146)
echoing b'Message 1 from client.Message 2 from client.' to ('127.0.0.1', 59144)
echoing b'Message 1 from client.Message 2 from client.' to ('127.0.0.1', 59146)
closing connection to ('127.0.0.1', 59144)
closing connection to ('127.0.0.1', 59146)
```

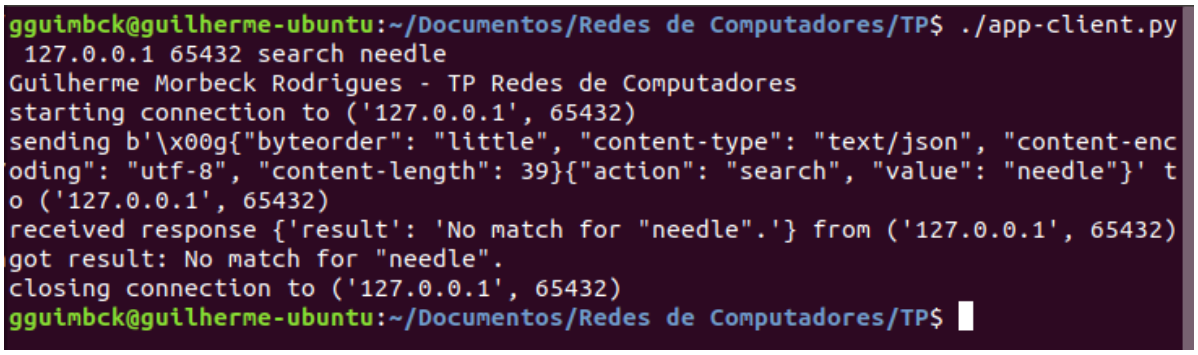
Após iniciar conexão, no terminal que está sendo executado o servidor é imprimido na tela as conexões que foram aceitas do cliente, a resposta ao cliente e o fechamento das conexões após as respostas. Nessa implementação, ao fim das conexões, o servidor continua no aguardo (listening) novas conexões.

Application Client and Server

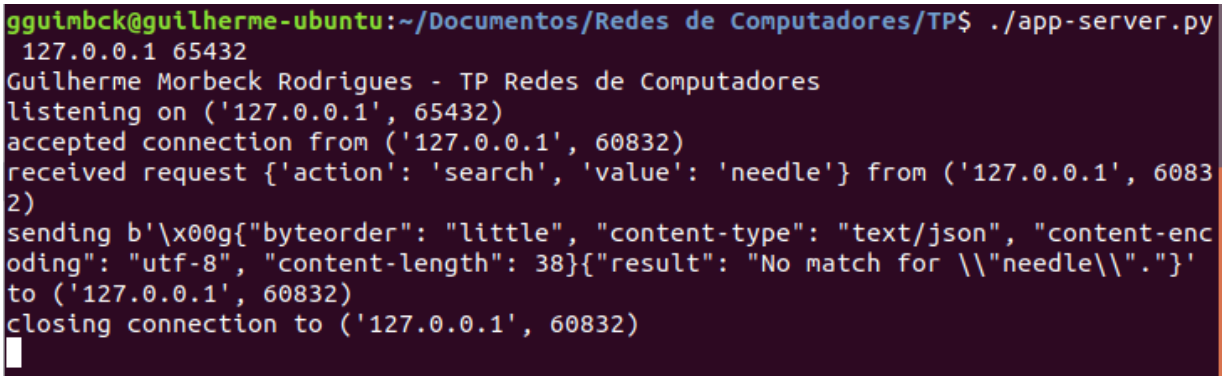
Os códigos app-server.py e app-client.py junto dos códigos libclient.py e libserver.py foram implementados com o objetivo de otimizar e tornar ainda mais real a aplicação do socket fazendo tratamento de exceções. Com isso, novas ações foram adicionadas na execução do código, mas a essência continua sendo a de uma múltipla conexão de cliente e servidor.

```
gguimbck@guilherme-ubuntu:~/Documentos/Redes de Computadores/TP$ ./app-server.py
 127.0.0.1 65432
Guilherme Morbeck Rodrigues - TP Redes de Computadores
listening on ('127.0.0.1', 65432)
```

Ao executar o app-server, o servidor é iniciado com a instanciação de um socket que fica no aguardo de uma conexão.



Executando o app-client, a conexão com o servidor é feita e, como previsto, o tratamento de exeções é realizado que não é conseguido o match, o valor passado como parâmetro não foi localizado. Mesmo sem o match, o cliente recebe a resposta do servidor de erro. Após isso a conexão é encerrada, mas não há uma quebra na execução do app-client.



Após a inicialização do app-client, é possível verificar na tela do terminal que executa o servidor que uma conexão foi aceita. O servidor recebe do cliente, que possui seu IP e porta, uma ação e um valor. Nesse caso, não foi encontrado o valor, o servidor trata o possível erro e envia a resposta ao cliente. Finalizando a conexão com o cliente e fica novamente ao aguardo de uma nova tentativa de conexão.