Exercício 3

NOME	RA
Renan Camargo de Castro	147775

Questão 1

- Função **5 - accept**:

A função accept extrai a primeira conexão disponível que chegar na fila de conexões pendentes.

Cria também um novo socket do mesmo tipo do passado como argumento, e aloca um novo file descriptor para aquele socket.

Declaração:

```c
int accept(int socket, struct sockaddr *restrict address,
       socklen_t *restrict address_len);
```

Onde:

- Socket é um descriptor de um socket que foi criado com socket(), associado à um endereço com bind() e fez uma chamada com sucesso à listen().
- address é um ponteiro para a estrutura sockaddr que contém que está conectando ao socket.
- address_len é ponteiro para o tamanho do endereço apontado pelo struct anterior.

Exemplo de uso:

```c
new_s = accept(s, (struct sockaddr *)&sin, &len)
```

Questão 2

Segue abaixo as saídas dos programas executados:

```Bash
➜  exercicio3 git:(master) ✗ ./server_antigo
Olá mundo!
Teste de saída, estou digitando no cliente!

➜  exercicio3 git:(master) ✗ ./client_antigo 127.0.0.1
Olá mundo!
Teste de saída, estou digitando no cliente!
```

Questão 3

Para diagnosticar o uso da rede, podemos usar o netstat, para confirmar se há sockets ativos e conexões estabelecidas: Execução:

```Bash
➜  exercicio3 git:(master) ✗ netstat
Active Internet connections
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
tcp4       0      0  192.168.1.107.51299    lhr08s06-in-f3.1.https FIN_WAIT_1
tcp4       0      0  192.168.1.107.51295    gru06s26-in-f14..https ESTABLISHED
tcp4       0      0  192.168.1.107.51293    gru09s18-in-f14..https ESTABLISHED
tcp4       0      0  192.168.1.107.51286    gru06s26-in-f14..https ESTABLISHED
tcp4       0      0  192.168.1.107.51285    gsademo28.google.imaps FIN_WAIT_1
**------------------------------------------------------------------------------
tcp4       0      0  localhost.31472        localhost.51283        ESTABLISHED
tcp4       0      0  localhost.51283        localhost.31472        ESTABLISHED
**------------------------------------------------------------------------------
tcp4       0      0  192.168.1.107.51276    gru06s25-in-f14..https ESTABLISHED
tcp4       0      0  192.168.1.107.51222    ce-in-f189.1e100.https ESTABLISHED
tcp4       0      0  192.168.1.107.51201    gru09s18-in-f3.1.https ESTABLISHED
tcp4       0      0  192.168.1.107.51195    gsademo28.google.imaps ESTABLISHED
tcp4       0      0  192.168.1.107.51133    gru06s26-in-f14..https ESTABLISHED
tcp4       0      0  192.168.1.107.50644    gsademo28.google.imaps ESTABLISHED
...
```

No caso, podemos observar uma conexão estabelecida entre as portas do cliente e do servidor. 31472/51283, o que indica atividade sobre a rede.

Podemos utilizar também o tcpdump sobre a interface de loopback lo0, para verificar os pacotes, que realmente estavam sendo enviados via TCP:

```Bash
➜  exercicio3 git:(master) ✗ sudo tcpdump -i lo0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on lo0, link-type NULL (BSD loopback), capture size 262144 bytes
22:55:57.127161 IP localhost.58615 > localhost.63342: Flags [.], ack 2919421103, win 12745,
22:55:57.127224 IP localhost.63342 > localhost.58615: Flags [.], ack 1, win 12736, options
22:55:57.656758 IP localhost.51283 > localhost.31472: Flags [P.], seq 3566560062:3566560069
22:55:57.656792 IP localhost.31472 > localhost.51283: Flags [.], ack 7, win 12756, options
22:55:58.182531 IP localhost.51664 > localhost.redwood-broker: Flags [S], seq 3564278694, w
22:55:58.182560 IP localhost.redwood-broker > localhost.51664: Flags [R.], seq 0, ack 35642
22:55:58.200708 IP localhost.51283 > localhost.31472: Flags [P.], seq 7:12, ack 1, win 1275
22:55:58.200742 IP localhost.31472 > localhost.51283: Flags [.], ack 12, win 12756, options
22:55:58.720696 IP localhost.51283 > localhost.31472: Flags [P.], seq 12:17, ack 1, win 127
22:55:58.720731 IP localhost.31472 > localhost.51283: Flags [.], ack 17, win 12756, options
22:56:00.939091 IP localhost.51666 > localhost.19536: Flags [S], seq 3902828028, win 65535,
22:56:00.939137 IP localhost.19536 > localhost.51666: Flags [R.], seq 0, ack 3902828029, wi
22:56:04.181715 IP localhost.51667 > localhost.redwood-broker: Flags [S], seq 907901644, wi
22:56:04.181756 IP localhost.redwood-broker > localhost.51667: Flags [R.], seq 0, ack 90790
22:56:04.628716 IP 192.168.1.107.netbios-ns > 192.168.1.255.netbios-ns: NBT UDP PACKET(137)
22:56:04.629009 IP 192.168.1.107.netbios-ns > 192.168.1.255.netbios-ns: NBT UDP PACKET(137)
22:56:04.629178 IP 192.168.1.107.netbios-ns > 192.168.1.255.netbios-ns: NBT UDP PACKET(137)
22:56:08.181445 IP localhost.51668 > localhost.redwood-broker: Flags [S], seq 4039994602, w
22:56:08.181491 IP localhost.redwood-broker > localhost.51668: Flags [R.], seq 0, ack 40399
22:56:08.841861 IP localhost.51670 > localhost.19536: Flags [S], seq 4178172786, win 65535,
22:56:08.841890 IP localhost.19536 > localhost.51670: Flags [R.], seq 0, ack 4178172787, wi
22:56:11.182096 IP localhost.51671 > localhost.redwood-broker: Flags [S], seq 519773658, wi
22:56:11.182136 IP localhost.redwood-broker > localhost.51671: Flags [R.], seq 0, ack 51977
22:56:12.181739 IP localhost.51672 > localhost.redwood-broker: Flags [S], seq 3950993065, w
22:56:12.181768 IP localhost.redwood-broker > localhost.51672: Flags [R.], seq 0, ack 39509
```

Embora tenham outros pacotes, é necessário prestar atenção aos que apresentam:

localhost.51283 > localhost.31472

ou vice-versa.

Questão 4

O telnet pode ser utilizado, pois é um programa que usa comunicação via sockets. Segue utilização:

```Bash
➜  exercicio3 git:(master) ✗ ./server_antigo
MEU TESTE
TESTE LAB DE REDES

➜  exercicio3 git:(master) ✗ telnet 127.0.0.1 31472
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
MEU TESTE
TESTE LAB DE REDES
```

Embora possa ser utilizado, há um problema com o tamanho do bufer(tamanho da mensagem enviada). No caso, o telnet envia somente a mensagem digitada no console para o server, e o mesmo imprime um pedaço de memória do tamanho da variável MAX_LINE(definida no código). Isso dá problema com o telnet, pois ele manda mensagens de tamanhos variados. Exemplo disso é o seguinte:

```Bash
➜  exercicio3 git:(master) ✗ telnet 127.0.0.1 31472
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
teste do telnet
teste do telnet
lixo
lixo

➜  exercicio3 git:(master) ✗ ./server
teste do telnet
lixo
do telnet
```

Vemos, que no caso, a frase “do telnet” após “lixo”, não era desejada. Isso decorre da mensagem ter sido enviada sem os caracteres zerados, isso é, sem o tamanho do buffer igualitário entre server e cliente. O server espera mensagens de tamanho fixo, mas o telnet envia com tamanhos variáveis, o que ocasiona o erro. Funciona quando as mensagens são enviadas menores para maiores, assim não tendo lixo no buffer. Mas quando são enviadas mensagens grandes, e depois uma mensagem menor, essa parte que ficou de lixo no buffer é impressa no server.

Questão 5

Segue amostra de uso:

Bash

```
➜  exercicio3 git:(master) ✗ ./client 127.0.0.1
teste
teste
lalalwelfawkefpwa kepfkawpoekfp awkefpowa kefpokaw epfk awpefk pawekfp awkefp awkefp wakefp
lalalwelfawkefpwa kepfkawpoekfp awkefpowa kefpokaw epfk awpefk pawekfp awkefp awkefp wakefp
apakwepfkawoefjaopw vjefpoaw jvefoawjnefpaw ipefojn apwoejfaowpefj opawjefop aweofp jawpoe
apakwepfkawoefjaopw vjefpoaw jvefoawjnefpaw ipefojn apwoejfaowpefj opawjefop aweofp jawpoe

➜  exercicio3 git:(master) ✗ ./server
teste
lalalwelfawkefpwa kepfkawpoekfp awkefpowa kefpokaw epfk awpefk pawekfp awkefp awkefp wakefp
apakwepfkawoefjaopw vjefpoaw jvefoawjnefpaw ipefojn apwoejfaowpefj opawjefop aweofp jawpoe
```