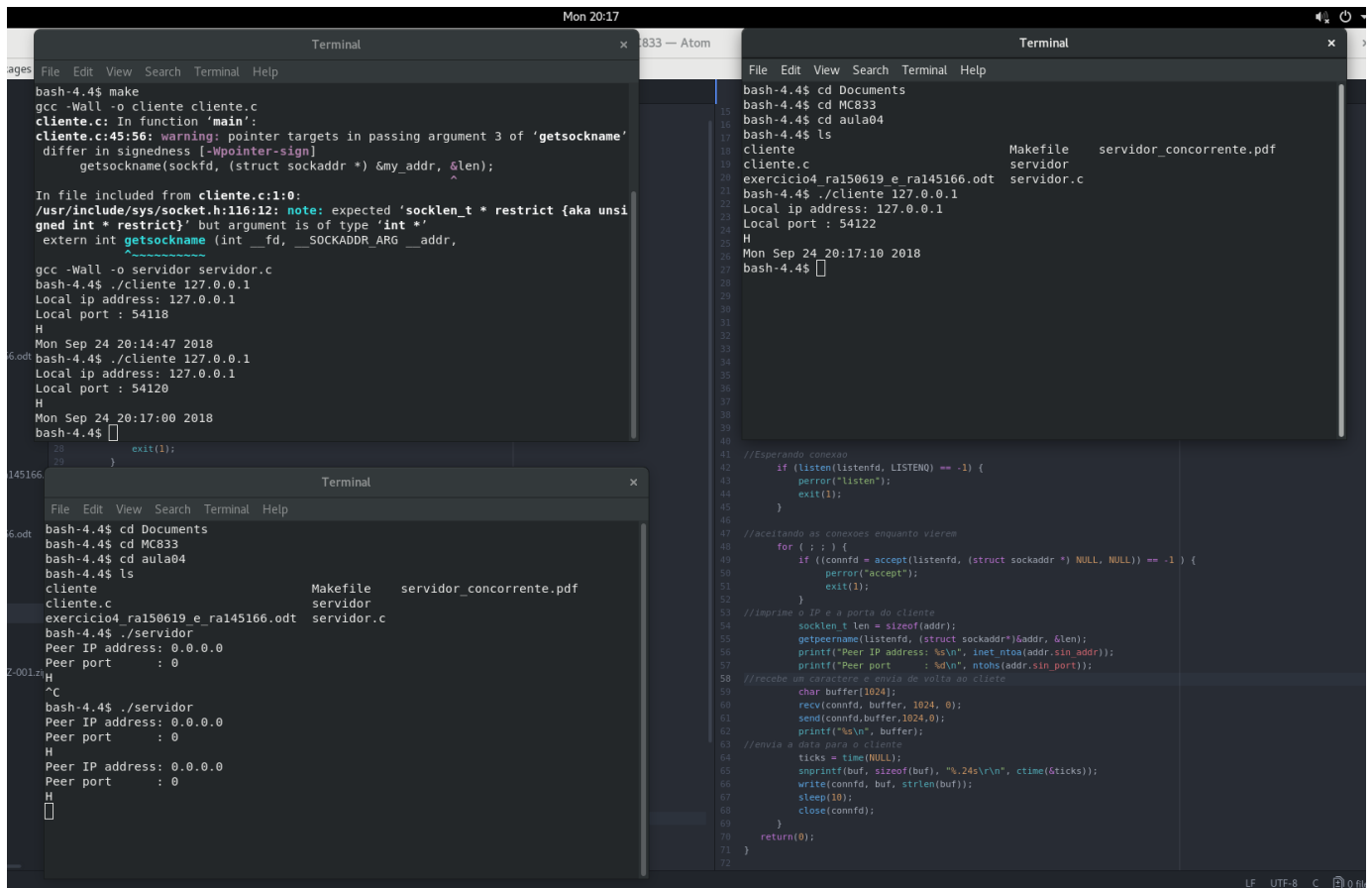


1) Adicione a função sleep no servidor.c da atividade prática anterior antes do socket connfd ser fechado de modo que o servidor "segure" a conexão do primeiro cliente que se conectar. Com essa modificação, o servidor aceita a conexão de dois clientes de forma concorrente? Por que? (Este código não precisa ser enviado por email)



```
bash-4.4$ make
gcc -Wall -o cliente cliente.c
cliente.c: In function 'main':
cliente.c:45:56: warning: pointer targets in passing argument 3 of 'getsockname'
differ in signedness [-Wpointer-sign]
    getsockname(sockfd, (struct sockaddr *) &my_addr, &len);
                                           ^
In file included from cliente.c:1:0:
/usr/include/sys/socket.h:116:12: note: expected 'socklen_t * restrict {aka unsigned int * restrict}' but argument is of type 'int *'
extern int getsockname(int __fd, __SOCKADDR_ARG __addr,
               ^
gcc -Wall -o servidor servidor.c
bash-4.4$ ./cliente 127.0.0.1
Local ip address: 127.0.0.1
Local port : 54118
H
Mon Sep 24 20:14:47 2018
bash-4.4$ ./cliente 127.0.0.1
Local ip address: 127.0.0.1
Local port : 54120
H
Mon Sep 24 20:17:00 2018
bash-4.4$

//Esperando conexao
if (listen(listenfd, LISTENQ) == -1) {
    perror("listen");
    exit(1);
}

//aceitando as conexoes enquanto vierem
for (;;) {
    if ((connfd = accept(listenfd, (struct sockaddr *) NULL, NULL)) == -1) {
        perror("accept");
        exit(1);
    }
}

//imprime o IP e a porta do cliente
socklen_t len = sizeof(addr);
getpeername(listenfd, (struct sockaddr*)&addr, &len);
printf("Peer IP address: %s\n", inet_ntoa(addr.sin_addr));
printf("Peer port : %d\n", ntohs(addr.sin_port));

//fecha um caractere e envia de volta ao cliente
char buffer[1024];
recv(connfd, buffer, 1024, 0);
send(connfd, buffer, 1024, 0);
printf("%s\n", buffer);

//envia a data para o cliente
ticks = time(NULL);
sprintf(buf, sizeof(buf), "%.24s\n", ctime(&ticks));
write(connfd, buf, strlen(buf));
sleep(10);
close(connfd);
return(0);
}
```

Como pode-se observar no data impressa por cada terminal cliente houve a espera de 10s no sleep do segundo executado. Não da para ser concorrente pois, ele está esperando uma conexão com o cliente, rodar exclusivamente o sleep no servidor não funciona, mas ao formar uma conexão ele só consegue se conectar novamente após ter a conexão atual encerrada.

4) No trecho de código abaixo, porque o servidor continua escutando e os clientes continuam com suas conexões estabelecidas mesmo após as chamadas dos close? Explique o porque do uso de cada close e se algum deles é dispensável neste trecho de código.

```
for (;;) {
    connfd = Accept (listenfd,...);

    if ( (pid=Fork()) == 0) {
        Close(listenfd);
        doit(connfd); // Faz alguma operação no socket
        Close(connfd);
        exit(0);
    }
```

```

    }
    Close(connfd);
}

```

Conforme dito no site da IBM:

Fontes: https://www.ibm.com/support/knowledgecenter/en/SSB23S_1.1.0.15/gtpc2/cpp_close.html
<https://softwareengineering.stackexchange.com/questions/195763/protocol-for-closing-a-socket-connection>

A função `Close` encerra o servidor associado a variável que o descreve, e libera os recursos alocados para ele. Se o socket se refere a uma conexão TCP aberta, a conexão é fechada. Se o socket está fechado quando há uma fila de dados de entrada, a conexão é reiniciada ao invés de simplesmente fechada.

Portanto para o encerramento garantido afim de tirar as duvidas par aa camada de aplicação devemos dar close nos 2 sockets, no cliente e no servidor, ambos são necessários para garantir o encerramento da conexão e liberação de memória da fila com segurança, já que um fica esperando os dados e o outros é responsável pela conexão em si.

5) No servidor concorrente, assuma que o processo filho é executado depois da chamada para `fork`. O filho então completa o serviço do cliente antes da chamada do `fork` retornar ao processo pai. O que acontece nas duas chamadas para `close`, no código da questão anterior?

In this piece of code:

```

if (pid == 0) {
    close(sockfd); // can't this cause problem ??
    dostuff(newsockfd);
    exit(0);
}

```

Fontes:

<https://stackoverflow.com/questions/5133865/parent-and-child-process-socket-close> <https://superuser.com/questions/1131778/closing-a-socket-which-keeps-waiting-a-child-process-when-the-parent-process-ha>

https://www.cs.ait.ac.th/~on/O/oreilly/perl/cookbook/ch17_10.htm

A função `close ()` afeta somente a cópia filha do descritor `sockfd`. A cópia-pai continua aberta para a próxima iteração do laço. É considerado boas maneiras de codificação que as cópias herdadas dos descritores dos processos filho sejam encerradas assim que não forem mais necessárias.

Quando o processo executa o `Fork`, o filho tem cópias de todos os filehandlers do pai, incluindo sockets. Quando o `Close` é acionado para um arquivo ou socket, fecha-se apenas a cópia de uso corrente, mas se outro processo, seja pai ou filho ainda possui uma cópia do socket em aberto, o sistema operacional não considera o socket fechado.

No caso em que o socket ainda troca dados, se os dois processos tem o socket aberto, um pode fechar, mas o sistema operacional ainda considera aberto, e isso pode gerar um deadlock ou uma confusão, enquanto o outro processo não encerra seu socket.

Para evitar isso, pode-se usar a função shutdown que é mais insistente que o close, ela diz para o sistema operacional que mesmo as cópias do filehandler deve ser fechadas e na outra ponta receber um EOF, caso estejam em leitura ou um SIGPIPE caso estejam em escrita.

6) Remova a chamada do bind, mas deixe disponível a chamada de listen. O que acontece? Explique.

Fonte: <https://stackoverflow.com/questions/741061/listen-without-calling-bind>

As chamdas funcionam, mas como não foi chamado o Bind explicitamente o sistema operacional ou a biblioteca implicitamente declara uma porta e fará o Bind, da mesma forma que dar Connect sem Bind. Para descobrir qual “nome” o sistema ligou ao socket, já que isso varia com o sistema operacional deve-se usar uma ferramenta como o netstat ou similar.

Por exemplo:

```
// Create socket and set it to listen (we ignore error handling for brevity)
int sock = socket(AF_INET, SOCK_STREAM, 0);
listen(sock, 10);

// Sometime later we want to know what port and IP our socket is listening on
socklen_t addr_size = sizeof(struct sockaddr_in);
struct sockaddr_in addr;
getsockname(sock, (struct sockaddr *)&addr, &addr_size);
```

7) Utilizando ferramentas do sistema operacional, qual dos lados da conexão fica no estado TIME_WAIT após o encerramento da conexão? Isso condiz com a implementação que foi realizada? Justifique.

Deve-se usar o Netstat para verificar isso, conforme diz a fonte:

<http://www.serverframework.com/asynchronevents/2011/01/time-wait-and-its-design-implications-for-protocols-and-scalable-servers.html>

O estado de TIME_WAIT geralmente é atingido por aquele que fez o Close ativamente primeiro, podendo ser o cliente ou o servidor, apesar de normalmente ser o cliente que realiza o Close. Como nós colocamos o função ShutDown no servidor quando o cliente dá o comando quit quem ficará nesse caso em time_wait é o servidor.