Introdução às Redes de Comunicação - 2014/15

Ficha de exercícios (TCP e IOMUX)

1. Desenvolva uma aplicação cliente-servidor em que os clientes enviam, via protocolo TCP, uma mensagem de texto, passada como argumento através da linha de comando, para um servidor cuja localização é igualmente passada na linha de comando. Estes ficam a aguardar por uma resposta. O servidor deve ir apresentado, na saída *standard*, as mensagens que vai recebendo e reenviar uma resposta cujo conteúdo corresponde ao tamanho destas em formato ascii.

2. Altere o código do cliente e do servidor desenvolvidos no âmbito do exercício anterior de modo a que estes sejam mais rigorosos no que se refere à utilização do protocolo TCP. Concretamente, pretende-se que o código reflicta o facto do protocolo TCP suportar fluxos de bytes em vez de fluxo de mensagens/datagramas. Para o efeito, desenvolva as funções *int* ***writeN****(SOCKET sock, char * buffer, int nbytes)*, que força o envio efectivo do número de bytes pretendidos, e *int* ***readLine****(SOCKET sock, char * buffer, int tamMax)*, que vai lendo bytes num *socket* TCP até encontrar o caractere '\n', atingir o tamanho máximo do buffer ou ocorrer uma situação de erro ou de fecho de ligação. Com a função *readLine*, os caracteres recebidos são colocados em *buffer*, sendo acrescentado o caractere '\0' no final de modo a terminar a *string* recebida. O caractere '\r' deve ser ignorado. Basicamente, pretende-se uma funcionalidade semelhante à das funções *fgets* ou *gets_s*, mas direcionada para uma ligação TCP. As funções *writeN* e *readLine* devolvem o número de bytes escritos/lidos, *SOCKET_ERROR* caso ocorra um erro ou zero caso a ligação tenha sido encerrada.

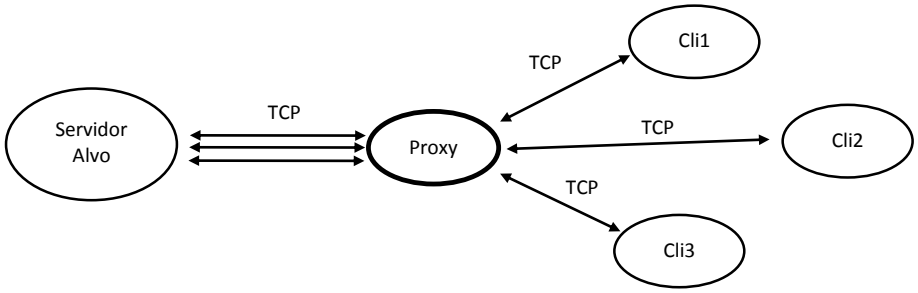
3. Altere o cliente desenvolvido de modo a que seja possível fornecer o nome do servidor (e.g., *smtp.isec.pt*) em vez do seu endereço IP (*193.127.78.34*).

4. Altere o servidor de modo a que este vá recebendo mensagens (i.e., sequências de caracteres terminadas em '\n') do cliente actual e enviando as respectivas confirmações até que seja recebida a sequência de caracteres "SAIR\n". Teste o servidor recorrendo a vários clientes Telnet conectados em simultâneo.

5. Altere o servidor desenvolvido de modo a que este passe a ser do tipo concorrente, ou seja, com capacidade para atender vários clientes em simultâneo. Recorra à função *getpeername* para obter a localização do par remoto associado a um determinado *socket* TCP conectado.

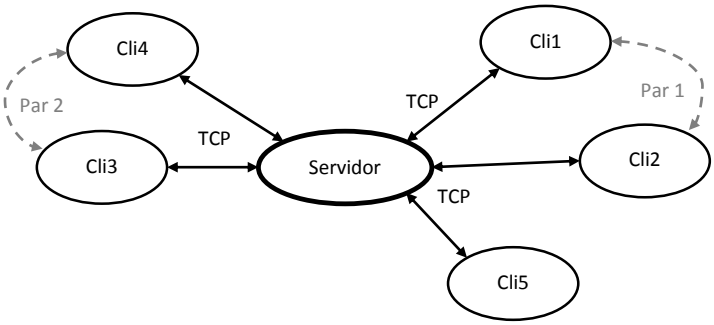
6. Altere o servidor desenvolvido de modo a que este passe, em simultâneo, a receber comandos da entrada *standard*. Quando é digitado o comando "sair", o servidor encerra o *socket* de escuta, deixando de aceitar novos clientes, e termina quando já não existirem clientes a serem atendidos.

7. Desenvolva um servidor TCP concorrente designado *proxy* e destinado a servir de ponte entre clientes e um servidor alvo específico cuja localização, à semelhança do porto de escuta pretendido, é passada através da linha de comando. Em concreto, para cada cliente ligado, o *proxy* estabelece uma ligação TCP com o servidor alvo e, a partir desse instante, envia ao cliente qualquer byte recebido deste último e vice-versa. Deste modo, os clientes passam a ter a ilusão de estarem directamente conectados ao servidor alvo apesar de terem estabelecido a ligação TCP com o *proxy*. Para efeitos de teste, recorra a clientes Telnet e utilize *pop.isec.pt:110* como servidor alvo.



8. Desenvolva um servidor TCP concorrente destinado a servir de ponte entre pares de clientes, sendo o porto de escuta pretendido passada através da linha de comando. Quando dois clientes se conectam ao servidor de forma sucessiva, este passa a enviar ao primeiro qualquer byte recebido do segundo e vice-versa. Deste modo, os dois clientes têm a ilusão

de estarem directamente conectados um ao outro apesar de terem estabelecido a ligação TCP com o servidor. Este repete a operação para os próximos dois clientes que se conectarem e assim sucessivamente.



9. Desenvolva uma versão mais genérica do servidor realizado no âmbito do exercício 8 em que são colocados em contacto grupos de clientes em vez de pares. Os grupos são formados com base em clientes que se conectam de forma sucessiva, ficando constituídos quando o limite MAX_TAM_GRUPO é atingido ou se aguarda mais de LIMIT_ESPERA segundos por uma nova ligação. Em cada grupo, qualquer byte recebido de um dos elementos é reencaminhado para os restantes. Quando um grupo passe estar formado, o servidor inicia a constituição do grupo seguinte continuando a fornecer o serviço de reencaminhamento de bytes aos grupos já existentes. Um grupo deixa de existir quando ocorre um problema qualquer ou quando um dos elementos deixa de estar conectado.

10. Desenvolva uma versão mais genérica do servidor realizado no âmbito do exercício 9 em que são suportadas baixas na constituição dos grupos, i.e., perdas de conectividade TCP com clientes devido a problemas/erros ou encerramentos. Um grupo deixa de existir apenas quando o número de clientes conectados é inferior a dois.

CRIAÇÃO, ASSOCIAÇÃO A UM PORTO LOCAL E FECHO DE SOCKETS WINDOWS

SOCKET **socket**(int af, int type, int protocol); /* PF_INET, SOCK_DGRAM, IPPROTO_UDP*/

int **bind**(SOCKET s, const struct sockaddr *name, int namelen);

int **closesocket**(SOCKET s);

ESTABELECIMENTO DE LIGAÇÕES TCP

int **connect**(SOCKET s, const struct sockaddr *name, int namelen);

SOCKET **accept**(SOCKET s, struct sockaddr *from, int *fromlen);

INDICAÇÃO DE ERRO E CÓDIGOS DE ERRO

SOCKET_ERROR

INVALID_SOCKET

int **WSAGetLastError**(void); /* WSAETIMEDOUT */

LOCALIZAÇÃO E CONVERSÃO DE FORMATOS

struct sockaddr_in a; /* a.**sin_family**, a.sin_addr.**s_addr**, a.**sin_port** */

…**htons**(…); /* host to network short */

…**htonl**(…); /* host to network long */

…**ntohs**(…); /* network to host short */

…**ntohl**(…); /* network to host long */

unsigned long **inet_addr**(const char *cp);

char* **inet_ntoa**(struct in_addr in); /* network to ascii */

RESOLUÇÃO DE NOMES

INADDR_NONE

struct hostent* **gethostbyname**(char *name);

/*

** **struct hostent** info;

** struct sockaddr_in addr;

** ...

** memcpy(&(addr.sin_addr.s_addr), info->**h_addr**, info->**h_length**);

**...

*/

ENVIO E RECEPÇÃO DE DATAGRAMAS

int **send**(SOCKET s, const char *buf, int len, int flags);

int **recv**(SOCKET s, char *buf, int len, int flags);

MULTIPLEXAGEM DE SOCKETS

int **select**(32, **fd_set** *readfds, NULL, NULL, **struct timeval** *timeout);

FD_ZERO(&set);

FD_SET(s, &set);

FD_ISSET(s, &set);

struct timeval { long tv_sec; long tv_usec;}

OBTENÇÃO DE INFORMAÇÃO LOCAL E REMOTA ASSOCIADA AOS SOCKETS

int **getpeername**(SOCKET s, struct sockaddr *name, int *namelen);

int **getsockname**(SOCKET s, struct sockaddr *name, int *namelen);

int **strcmp**(const char *s1, const char *s2);

char * **strcpy_s**(char * strDestination, int sizeStrDestination, const char * strSource);

CONFIGURAÇÃO DE OPÇÕES/PARÂMETROS

int **setsockopt**(SOCKET s, int level, int optname, const char *optval, int optlen);

/* level = SOL_SOCKET, optname = SO_RCVTIMEO, optval = (char *)&timeoutMsec (DWORD ** timeoutMsec;) */

CRIAÇÃO DE THREADS

HANDLE WINAPI **CreateThread**(

_In_opt_ LPSECURITY_ATTRIBUTES **lpThreadAttributes**,

_In_ SIZE_T dwStackSize,

_In_ LPTHREAD_START_ROUTINE **lpStartAddress**,

_In_opt_ LPVOID **lpParameter**,

_In_ DWORD dwCreationFlags,

_Out_opt_ LPDWORD **lpThreadId**

);

/*

** void **AtendeCliente**(LPVOID param);

** SECURITY_ATTRIBUTES **sa**;

** DWORD **thread_id**;

** ...

** sa.nLength=sizeof(sa);

** sa.lpSecurityDescriptor=NULL;

** ...

** **CreateThread**(&**sa**, 0, (LPTHREAD_START_ROUTINE)**AtendeCliente**, (LPVOID)param,

** (DWORD)0, &**thread_id**);

*/

void **ExitTread**(…);