Exercício 7 de MC833 — Programação em Redes de Computadores

Raul Rabelo Carvalho, 105607, turma A

22 de Maio de 2014

1 Decisões de projeto

Para seguir a instrução do Exercício 7 de empregar *wrappers* para todas as chamadas de função, foi empregado um arquivo *header* comum tanto ao servidor quanto ao cliente contendo as configurações, tipos, macros e pragmas das funções *wrapper* dos dois *softwares*. Além deste arquivo, dois outros arquivos compõem o sistema cliente/servidor: um arquivo contendo a implementação dos *wrappers* relacionados à rede e um segundo arquivo no qual são implementados todos as outras funções usadas.

O cliente foi implementado para receber entradas do teclado e, mesmo no modo UDP, ele utiliza a função `Connect` para simplificar a implementação.

O servidor é concorrente e cria processos-filhos para atender a conexões TCP.

1.1 Manual de uso

O sistema é compilado com o comando "make", quando o `Makefile` suprido não foi alterado. "make clean" está disponível para remover os arquivos binários.

O cliente deve ser executado com a seguinte linha de comando "./cliente PROTOCOLO ENDERECO PORTA", sendo que o último argumento é opcional (quando a porta não é passada, a porta assumida é a 49151).

O servidor deve ser executado com a seguinte linha de comando "./servidor PORTA", sendo a porta é opcional (quando a porta não é passada, a porta assumida é a 49151).

2 Detalhes de implementação

Primeiramente, as funções *wrapper* usadas em programação de sockets na linguagem C do livro-texto foram alteradas para usar a função `perror` como saída de erro; fora isso, nada mais foi alterado.

No arquivo `auxf.c` foram implementadas as funções usadas para tratar os argumentos do cliente e do servidor (cliArgs e srvArgs, respectivamente), coletando os dados para estabelecimento da conexao: somente a porta, no caso do servidor; e protocolo, endereço IP e porta, no caso do cliente. Estes dados são usados para preencher a estrutura `sockaddr` em cada um dos programas. Além dessas duas funções, `auxf.c` contém as duas funções para criação de processos-filhos e para o tratamento de processos inativos — `Fork()` e `signalHandler()` respectivamente. Finalmente, há uma função de comparação que testa se uma *string* contém a palavra "exit"; esta função é usada tanto no cliente quanto no servidor para encerrar a conexão graciosamente.

O arquivo *header* `myNetworking.h` é simples, contendo somente fazendo a inclusão das bibliotecas necessárias, definido algumas macros de configuração e os dois tipos empregados no sistema. O primeiro tipo é somente um booleano; e o segundo enumera os protocolos de transporte (TCP e UDP).

2.1 ServEco.c

O servidor de eco tem uma implementação simples e direta devido ao uso de funções *wrapper*: são criados e inicializados dois *sockets*, um do protocolo TCP e orientado a conexões e outro do protocolo UDP e orientado a mensagens. Usando a funçao `Select`, o servidor continuamente testa se há alguma conexão a ser estabelecida ou mensagem recebida nos dois *sockets* e, quando for o caso, trata o evento.

Caso o *socket* TCP tenha uma conexão para ser estabelecida, a conexão é aceita e uma cópia do processo é criada para atender ao cliente que pediu a conexão. O processo-filho entra em *loop* recebendo e re-enviando as mensagens do cliente até ser encerrado. O processo-pai simplesmente volta ao `Select`.

Caso o *socket* UDP tenha uma mensagem para ser recebida, após recebê-la, o servidor a re-envia e retorna ao `Select`.

2.2 Cliente.c

A implementação de uma cliente TCP/UDP é extremamente facilitada pelo fato de mesmo um cliente UDP poder estabelecer uma conexão e enviar e receber mensagens usando as mesmas funções que um cliente TCP. O único ponto deste cliente que difere de um cliente puramente TCP é o momento em que o *socket* é criado: a depender do protocolo escolhido pelo usuário, o *socket* é criado como TCP ou UDP. De resto, o cliente realiza um *loop* no qual o teclado é lido, o que é lido é enviado ao servidor e o eco do servidor é impresso na saída padrão.

Tal como no caso do servidor, o código do cliente ficou extremamente enxuto devido o uso dos *wrapper*: a remoção dos testes para tratamento de erro do código principal tornou o código mais legível.

3 Testes