

# 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 conexão: 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 função **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

O sistema foi testado executando-se o servidor e dois clientes, como mostrado na figura 1. Cada um dos clientes conectou-se ao servidor usando um protocolo diferente e cada um enviou duas mensagens. Além disso, a figura ?? também inclui o programa htop mostrando o processo-filho que está atendendo ao cliente TCP.

Além do teste da figura. Foram testados múltiplos clientes TCP e múltiplos clientes UDP com sucesso. Finalmente, um último teste entre um laptop rodando ArchLinux e uma máquina virtual Linux Mint foi feito em uma rede local (LAN), também com sucesso.

Figura 1: Teste do sistema.

```
File Edit View Bookmarks Settings Help
rrcarvalho@Tarkin ~/MC833/exercicio7 $ make
gcc -Wall -c auxf.c
gcc -Wall -c wrapsock.c
gcc -Wall -o servidor auxf.o wrapsock.o serveco.c
gcc -Wall -o cliente auxf.o wrapsock.o cliente.c
rm -f auxf.o wrapsock.o
rrcarvalho@Tarkin ~/MC833/exercicio7 $ ./servidor
[]

exercicio7 : cliente - Konsole
File Edit View Bookmarks Settings Help
rrcarvalho@Tarkin ~/MC833/exercicio7 $ ./cliente UDP 127.0.0.1
Oi, servidor.
Oi, servidor.
Sou UDP.
Sou UDP.
[]

exercicio7 : cliente - Konsole <2>
File Edit View Bookmarks Settings Help
rrcarvalho@Tarkin ~/MC833/exercicio7 $ ./cliente tcp 127.0.0.1
Hello, server. I'm TCP.
Hello, server. I'm TCP.
See ya!
See ya!
[]

rrcarvalho : htop - Konsole
File Edit View Bookmarks Settings Help

 1  [|||||]          7.9%]   Tasks: 76, 136 thr; 3 running
 2  [||]            6.0%]   Load average: 1.55 0.98 0.58
 3  [|||||]         100.0%]  Uptime: 2 days, 16:19:51
 4  [|||]           6.0%]
Mem[|||||]         932/5860MB]
Swp[|]             0/7167MB]

PID USER      PRI  NI  VIRT   RES   SHR  S  CPU% MEM%   TIME+  Command
 1 root         20   0 25732  3316  1828  S   0.0  0.1   0:03.93 /sbin/init \EFI\gr
9178 rrcarvalh    20   0 547M  41120 22112 S   1.3  0.7   0:08.34 kdeinit4: kons
9392 rrcarvalh    20   0 15780  2340  1708  S   0.0  0.0   0:00.00 /bin/bash
9399 rrcarvalh    20   0 14132  2184  1324  R   0.7  0.0   0:00.75 htop
9281 rrcarvalh    20   0 15904  2388  1744  S   0.0  0.0   0:00.02 /bin/bash
9375 rrcarvalh    20   0 4196   332   268  S   0.0  0.0   0:00.00 ./client
9221 rrcarvalh    20   0 15904  2392  1748  S   0.0  0.0   0:00.01 /bin/bash
9380 rrcarvalh    20   0 4196   336   268  S   0.0  0.0   0:00.00 ./client
9180 rrcarvalh    20   0 15904  2392  1748  S   0.0  0.0   0:00.03 /bin/bash
9369 rrcarvalh    20   0 4056   336   268  R  100.0  0.0  3:04.77 ./servid
9381 rrcarvalh    20   0 4056    80    0  S   0.0  0.0   0:00.00 ./ser
9179 rrcarvalh    20   0 547M  41120 22112 S   0.0  0.7   0:00.00 kdeinit4: k
```