MC833 - Projeto 2

Carlos Avelar (168605) e Tiago Loureiro Chaves (187690)

09 de Maio, 2019

1 Introdução

Este projeto teve como objetivo a implementação e comparação de um sistema cliente-servidor utilizando sockets, com comunicação baseada nos protocolos UDP e TCP, na linguagem de programação C.

O sistema baseia-se em uma rede de perfis profissionais com opções de consulta a dados oferecidas pelo servidor, os quais então são mostrados ao cliente. O cliente pode requisitar opções (p.e. listagem de um perfil, listagem de todos os perfis, etc.) até que decida finalizar a comunicação.

Assim, os dados enviados do cliente para o servidor são exclusivamente textuais (opção requisitada), enquanto que o servidor, além de texto, também pode enviar arquivos de imagem `.png` (fotos dos perfis).

Por fim, realizou-se uma análise do tempo médio de comunicação das opções, considerando tanto seu tempo de execução, no servidor, como o tempo total observado no cliente, e comparou-se os resultados entre as implementações UDP e TCP.

2 Sistema UDP

Ao iniciar-se o servidor (executando-se `./server` após gerados os arquivos executáveis), é estabelecida a ligação (*bind*) com a porta 3490 para escutar por mensagens de clientes.

Analogamente, iniciamos o cliente com `./cliente <hostname>`, onde `hostname` é o nome ou endereço da máquina onde foi executado o servidor (p.e. `localhost` se ambos estiverem rodando na mesma).

Em seguida, o cliente imprime para o usuário a seguinte lista de opções disponíveis:

- (1) dado o email de um perfil, retornar nome, sobrenome e foto;
- (2) dado o email de um perfil, retornar suas informações;
- (3) listar todas as informações de todos os perfis;
- (4) sair.

Considera-se então que o usuário digita corretamente o número da opção que deseja, e o email do perfil para as opções 1 e 2, e o cliente então envia as informações necessárias ao servidor em um datagrama UDP.

Ao receber mensagens [1], o servidor consulta o banco de dados (`db`) e envia o resultado de volta ao cliente, que tenta receber datagramas durante `TIMEOUT_IN_S` segundos (constante definida em

`client.c`) usando a função `poll()` [2]. Caso a resposta não seja recebida pelo cliente dentro desse intervalo de tempo, algo que devemos levar em conta uma vez que o protocolo UDP não garante a entrega de pacotes, a seguinte mensagem é mostrada: *"O servidor não respondeu a tempo (timeout de `TIMEOUT_IN_S` atingido)"*, com o nome da constante substituído pelo seu valor em segundos.

Já caso a resposta do servidor seja recebida, os resultados são *printados* no terminal do cliente, e salvos no arquivo `option_results.txt`, que mantém um registro de todas as opções selecionadas (imagens são salvas como `<email>.png`, onde `email` refere-se ao perfil dono da foto, e é a chave da tabela no `db`).

As opções disponíveis são novamente mostradas ao cliente, e repete-se o fluxo descrito acima, até que o usuário escolha sair (opção 4).

3 Armazenamento e estrutura de dados do servidor UDP

Assim como no Projeto 1 ([3]), usou-se um banco de dados `SQLite` para o armazenamento dos dados pela naturalidade apresentada na consulta de informações dos perfis com o uso de *queries* SQL.

Para isso, ao iniciar-se o servidor é chamada a função `init_db()` que cria as tabelas necessárias, mostradas abaixo, e as popula com informações:

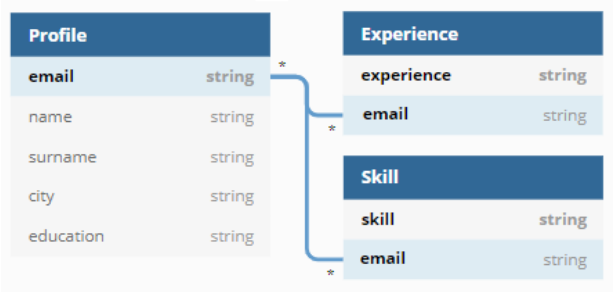

Figura 1: Tabelas do bando de dados

Os dados inseridos são os seguintes (email, nome, sobrenome, cidade, formação, habilidades e experiências):

cinco@mail.com	tres@mail.com	uno@mail.com
Cinco	Tres	Uno
Seis	Cuatro	Dos
Seattle	Campinas	Campinas
CS	CS	Linguistics
English, Reap, Sow, Spanish	Code, Read, Write	Acoustic Engineering, English
Study, Study more, Work	Study, Work	Research, Work

Nota-se que as fotos dos perfis não são guardadas no banco de dados, mas sim como imagens em uma pasta `imgs/` no mesmo diretório do arquivo executável do servidor, nomeadas `<email>.com`, sendo `email` a chave do perfil no `db` (evitando conflitos de nomenclatura).

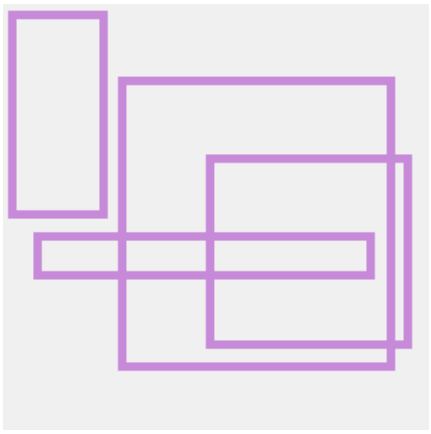

Figura 2: cinco@mail.com.png

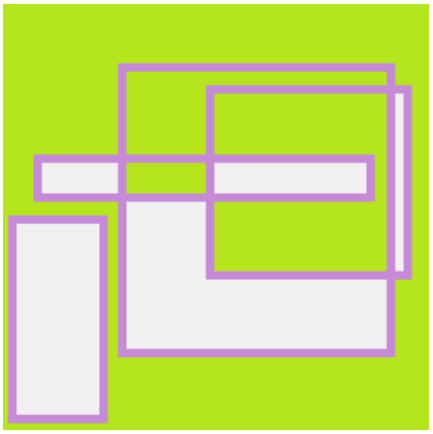

Figura 3: tres@mail.com.png

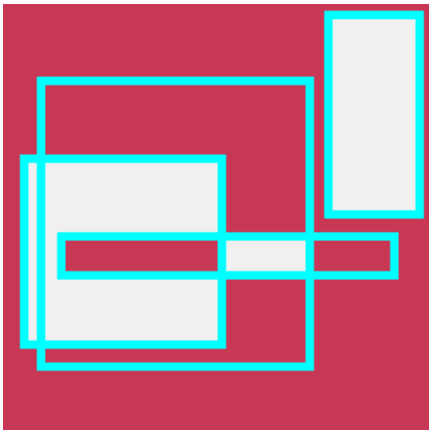

Figura 4: uno@mail.com.png

4 Detalhes da implementação do servidor UDP

Ao iniciar, o servidor cria as tabelas da Figura 1 no `db` (adicionando a elas as informações mencionadas acima). Para isso, inicializa-se o banco de dados com a função `sqlite3_open()` [4] e executam-se *queries* com `sqlite3_exec()` [5], a partir de chamadas à função `execute_sql()` do servidor. Nota-se que o resultado da última *query* realizada é sempre salvo no arquivo `last_query_result.txt` para referência.

Em seguida, o servidor cria uma `struct addrinfo` referente ao seu endereço IP chamando a função `getaddrinfo()` [6], com porta 3490 (definida pela constante `PORT` em `libs.h`), e socket do tipo `SOCK_DGRAM`, como queremos que a comunicação seja pelo protocolo UDP.

O servidor faz então a ligação (*bind*) de um socket com o endereço especificado pela `struct addrinfo` e entra em um *loop* para receber pacotes enviados para o socket criado.

Diferente do servidor TCP (cf. [3]), não é necessária a chamada de `listen()` ou `accept()` pois o protocolo UDP não mantém conexão (ele é *connectionless* e *stateless*).

Além disso, como o servidor UDP é iterativo, e não concorrente (como o servidor TCP implementado), cada mensagem recebida é tratada pelo mesmo processo, ao invés de ser feito um `fork()` para cada uma delas.

Ao receber uma mensagem de um cliente, é invocada a função relativa à opção escolhida, dentre as listadas na Seção 2. São essas:

- (1) `opt_get_full_name_and_picture_from_profile()`
- (2) `opt_get_profile()`
- (3) `opt_get_profiles()`

Como o protocolo UDP é *connectionless*, o processo cliente não precisa enviar a opção 4 (sair) para o servidor.

Após lida a mensagem do cliente, as funções acima executam as consultas necessárias no `db` (encapsuladas por funções de mesmo nome, porém com o sufixo `_sql`, que chamam `execute_sql()`). Por fim, com o resultado da *query* salvo em `last_query_result.txt`, executa-se `sendto_file_to_client()` para enviar seu conteúdo ao cliente, e `sendto_picture_to_client()` para enviar as fotos de perfil (salvas na pasta `/imgs` do mesmo diretório de `server.c`).

Já no cliente, se a resposta do servidor é obtida antes do timeout determinado para evitar a espera infinita por pacotes perdidos, os dados textuais são impressos na tela e as fotos são salvas (com o mesmo nome que no servidor, i.e. `<email>.png`, veja `save_img()`). Para referência, todas as respostas de consultas são anexadas no arquivo `option_results.txt`, chamando-se `save_result_to_file()`.

4.1 *Wrappers* de mensagens

Para tornar a comunicação mais fácil e abstrair o tratamento das chamadas de funções de troca de mensagens UDP, foram criados *wrappers* para o envio e recebimento de mensagens, usados no cliente e servidor. Na figura abaixo estão listados os *wrappers*, implementados em `common.c`:

```
int sendto_wrapper(int file_descriptor, char *message,
                   const struct sockaddr *dest_addr, socklen_t addrlen);
int recvfrom_wrapper(int file_descriptor, char **buffer,
                     struct sockaddr *src_addr, socklen_t *addrlen);
int recvfrom_wrapper_timeout(int file_descriptor, char **buffer,
                             struct sockaddr *src_addr, socklen_t *addrlen, int timeout_in_s);

int sendto_img_wrapper(int file_descriptor, char * message, int msg_size,
                       const struct sockaddr *dest_addr, socklen_t addrlen);
int recvfrom_img_wrapper(int file_descriptor, char **buffer, int *size,
                         struct sockaddr *src_addr, socklen_t *addrlen);
int recvfrom_img_wrapper_timeout(int file_descriptor, char **buffer, int *size,
                                 struct sockaddr *src_addr, socklen_t *addrlen, int timeout_in_s);
```

Figura 5: *Wrappers* para `sendto()` e `recvfrom()`

Como o protocolo UDP não mantém conexão, é preciso especificar para cada pacote enviado o seu endereço de destino [7], assim como é preciso especificar o endereço de origem de um pacote ao tentar recebê-lo.

Logo, diferente dos *wrappers* implementados para o protocolo TCP (cf. [3], seção 4.1), ao utilizarmos o protocolo UDP é necessário um parâmetro extra, o endereço para o qual se deseja mandar ou receber a mensagem (`dest_addr` e `src_addr`).

Além disso, duas funções novas foram adicionadas (com sufixo `_timeout`) para que a espera por mensagens seja abortada após um período de tempo. Elas são usadas pelo cliente, pois é possível que a resposta do servidor se perca como o protocolo UDP não garante o recebimento de mensagens, e caso não houvesse um timeout, o cliente ficaria para sempre bloqueado esperando por uma resposta.

Como mencionado na Seção 2, utilizou-se a função `poll()` para verificar se há novas mensagens durante o período determinado. Optou-se pelo uso de `poll()` ao invés da função `select()` pois ele é mais atual e sua assinatura é mais simples.

5 Resultados obtidos

Executando os sistemas cliente-servidor UDP e TCP em máquinas distintas, porém na mesma rede (no laboratório CC00 do IC3), marcou-se 50 vezes para as opções 1 e 2 os tempos total (visto pelo cliente) e de processamento no servidor, a partir dos quais podemos estimar o tempo de comunicação, como mostra a Figura 6.

Dessa forma podemos avaliar se uma diferença pequena no tamanho da mensagem tem grande impacto no tempo levado ao utilizar-se UDP ou TCP. A opção 3 foi adicionada para que o usuário pudesse consultar quais perfis estão no banco, sendo que uma análise mais detalhada dela, assim como de outras opções, é feita em [3] (cf. Seção 5).

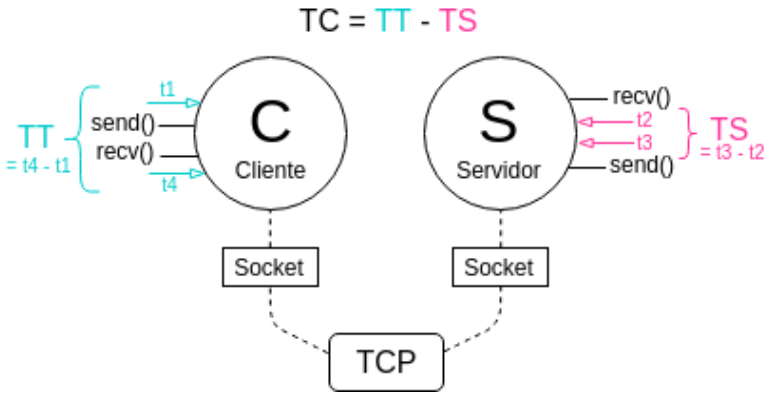

Figura 6: Tempos de comunicação (TC), total (TT) e de processamento no servidor (TS)

As tabelas abaixo mostram os resultados do experimento, sendo *Avg Com. Time* o tempo de comunicação (TC) e *Avg Op. Time* o tempo de processamento por opção no servidor (TS), em micro segundos (μs). O desvio padrão e intervalo de 95% de confiança são referentes ao TC:

Op.	Avg Com. Time (μs)	Std Deviation (μs)	95% Conf Interval (μs)	Avg Op. Time (μs)
1	85758,38	746,60	84265,17 to 87251,59	5342,83
2	85877,66	711,45	84454,75 to 87300,56	5623,44

Tabela 1: Valores para o cliente-servidor TCP.

Op.	Avg Com. Time (μs)	Std Deviation (μs)	95% Conf Interval (μs)	Avg Op. Time (μs)
1	5772,95	420,35	4932,25 to 6613,64	4703,08
2	5536,78	439,68	4657,41 to 6416,14	4910,94

Tabela 2: Valores para o cliente-servidor UDP.

Apresenta-se nos gráficos seguintes os tempos de comunicação calculados, a média das execuções (em vermelho), e barras delimitando o intervalo com nível de 95% de confiança.

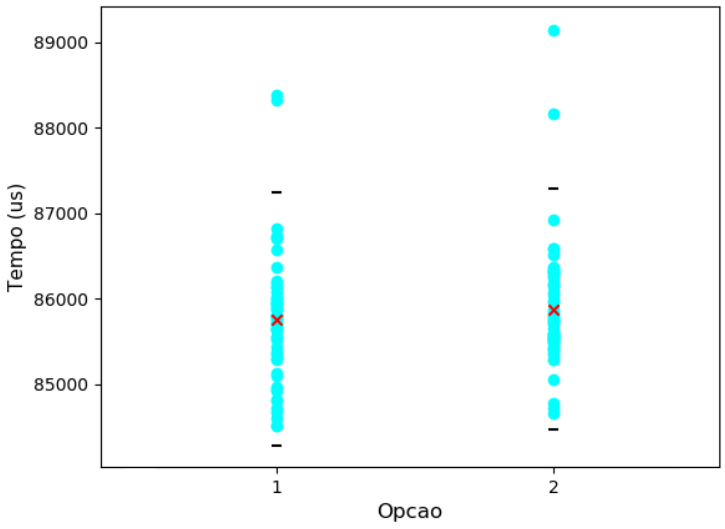

Figura 7: Tempos de comunicação (TC) por opção, com o protocolo TCP

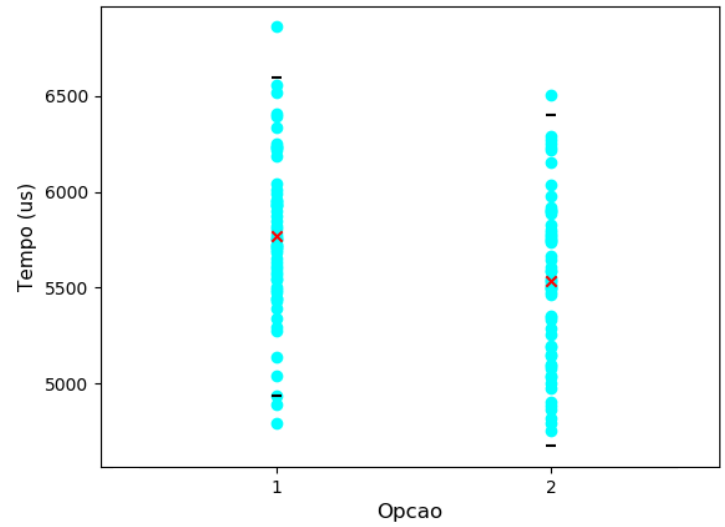

Figura 8: Tempos de comunicação (TC) por opção, com o protocolo UDP

6 Comparação das tecnologias (TCP × UDP)

Como esperado, vemos pelos Gráficos 7 e 8 que o tempo de comunicação ao utilizar-se o protocolo TCP foi bem maior que o UDP, apresentando uma ordem de grandeza de diferença.

Apesar de pequenas variações que podem ocorrer ao executar-se os programas sequencialmente, vemos pelas Tabelas 1 e 2 que o tempo de processamento no servidor foi próximo (com variação de apenas 10%), também conforme aguardado, pois o processamento (i.e. a consulta ao `db`) ocorre da mesma forma nas duas aplicações cliente-servidor.

Logo, podemos assim atribuir a variação no TC à diferença no protocolo utilizado.

Além disso, conforme mencionado na Seção 5, a diferença entre as opções 1 e 2 é de poucos

bytes, comparado com o tamanho total das mensagens. Observamos então que uma mudança pequena em relação ao tamanho da mensagem enviada não aparenta levar a nenhuma mudança significativa no tempo de comunicação, tanto no protocolo TCP como no UDP, ao analisarmos os intervalos de confiança em conjunto com a média dos Gráficos 7 e 8.

A Tabela 3 mostra uma visão simplificada das diferenças entre os protocolos.

	TCP	**UDP**
Inconveniência na implementação	Inicialização do servidor (conexão)	Recebimento de mensagens (timeout)
Confiabilidade	Garantida	Não confiável (depende da qualidade do meio)
Nível de abstração	Orientado a fluxo de bytes	Orientado a mensagens
Tempo de comunicação	Lento	Rápido

Tabela 3: Comparação entre os protocolos UDP e TCP.

7 Conclusão

Os dois protocolos são úteis em para diferentes tipos de aplicações, no caso da aplicação desenvolvida, o TCP parecia mais adequado por ser um protocolo confiável, como velocidade não era um problema devido ao pequeno tamanho dos arquivos utilizados. Caso o sistema desenvolvido fosse um pequeno serviço de *streaming*, um chat, ou um jogo *multiplayer* simples, o UDP provavelmente faria mais sentido, pela sua natureza *best-effort* e pelo fato dos sistemas listados tolerarem perdas pequenas.

O TCP pode parecer mais complicado de implementar em primeira análise devido a sua inicialização mais extensa, mas na hora de troca de mensagens o UDP requer um pouco mais de esforço para o tratamento das possíveis perdas (quando isso é necessário), e necessita de mais parâmetros para cada mensagem por ser *stateless*. No fim das contas, a implementação de ambos sistemas é praticamente equivalente em tamanho de código e em complexidade.

Um ponto positivo do UDP é a orientação à mensagem, que facilita o envio de mensagens de tamanho relativamente pequeno (menor do que 64Kb), pois não é necessário nenhum tratamento para diferenciar uma mensagem de outra, uma vez que cada uma é enviada completa e isoladamente. Já no TCP, como temos o conceito de fluxo de bytes, é necessário um *loop* para garantir que todos os bytes da mensagem sejam enviados, além de alguma forma de reconstruí-la e separá-la no destino.

Por outro lado, o UDP não garante a entrega da mensagem, o que leva a um *trade-off* interessante, pois não há assim uma "melhor opção", é necessário levar em conta as especificidades de cada aplicação. Por exemplo, em redes locais cabeadas (como foi o cenário testado) é vantajoso utilizar o UDP, pois ele apresentará velocidades maiores e com uma perda muito baixa, não gastando muito tempo em reenvios quando necessário.

Referências

[1] "recvfrom()." [Online]. Available: https://linux.die.net/man/2/recvfrom

[2] "poll()." [Online]. Available: https://beej.us/guide/bgnet/html/multi/pollman.html

[3] C. Avelar and T. Chaves, “MC833 - Projeto 1.” [Online]. Available: https://github.com/laurelkeys/computer-networks/blob/master/reports/relatorio_projeto1.pdf

[4] “Opening a new database connection.” [Online]. Available: https://www.sqlite.org/c3ref/open.html

[5] “One-step query execution interface.” [Online]. Available: https://www.sqlite.org/c3ref/exec.html

[6] “getaddrinfo().” [Online]. Available: http://man7.org/linux/man-pages/man3/gai_strerror.3.html

[7] “send(), sendto().” [Online]. Available: https://beej.us/guide/bgnet/html/multi/sendman.html