RELATÓRIO PROJETO 2
MC833 - LAB. DE REDES
1s2019

NOME: João Paulo Soubihe

RA: 151106

- ## Introdução

Continuamos nosso aprendizado com relação a protocolos de redes de computadores com a implementação de um servidor UDP e um servidor TCP, além de seus respectivos clientes. Ambos realizam a mesma tarefa, retornar ao cliente o nome, sobrenome e uma imagem da sua conta. Ainda em C, avaliamos cada detalhe da comunicação cliente-servidor, a seguir neste relatório descreveremos a sua execução e compararemos características dos dois protocolos.

- ## Sistema

Nosso sistema foi simplificado e executará apenas uma funcionalidade. No caso do servidor UDP,

1. UDP

Server

```
listener: waiting to recvfrom...
listener: got packet from 143.106.16.49
listener: packet is 13 bytes long
listener: packet contains "ana@gmail.com"
login de ana@gmail.com.txt
listener: sent 11 bytes to client
```

Client

```
talker: sent 13 bytes to 143.106.16.49
talker: waiting to recvfrom...
Bem Vindo!!
Ana Rocha
```

2. TCP

Server

```
server: got connection from 127.0.0.1
login de ze@gmail.com.txt
```

Client

```
client: connecting to 127.0.0.1
Bem-vindo!
Login:
ze@gmail.com
(0) Sair
(1) Dado o email de um perfil, retornar nome, sobrenome e foto da pessoa.
1
Jose Cardoso
```

A ideia desse projeto continua em concentrar todos os perfis e informações sobre eles em um servidor TCP ou UDP que proverá dados aos clientes que se conectarem a eles, de acordo com o protocolo adotado.

- **<u>Armazenamento</u>**

Esses perfis se encontrarão no server, na forma de arquivos .txt e uma imagem de perfil em .jpeg. Ambos os programas do tipo servidor se localizarão na pasta *Server* e ambos os clientes na pasta *Client*. Assim temos menor redundância de arquivos, mas ao mesmo tempo devemos prestar atenção em qual servidor está sendo executado de acordo com o tipo de conexão que desejamos.

Tais arquivos estarão organizados da seguinte forma:

➔ Arquivos .txt

Estarão reunidos na mesma pasta do programa servidor e terão nomes correspondentes ao seu email como EMAIL.txt organizados como no exemplo abaixo

```
Email: ana@gmail.com
Nome: Ana Sobrenome: Rocha
Residencia: Valinhos
Formacao Academica: Engenharia de Alimentos
Habilidades: Análise de Dados, Internet das Coisas, Computação em Nuvem
Experiência: (1) Estágio de 1 ano na Empresa X, onde trabalhei como analista de dados
(2) Trabalhei com IoT e Computação em Nuvem por 5 anos na Empresa Y
(3) Curso de Office avançado
```

Figura 1 - Modelo de perfil do cliente

➔ Arquivos .jpg
Também serão armazenados na pasta do programa servidor, com a identificação EMAIL.jpg

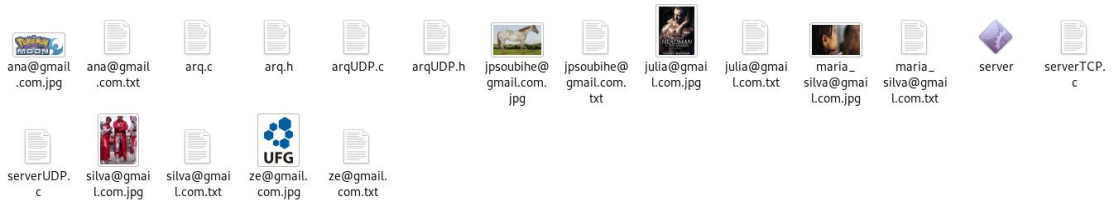

Figura 2 - Organização da pasta Server

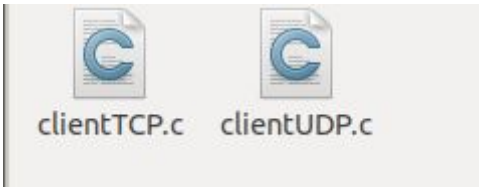

Figura 3 - Organização da pasta Client

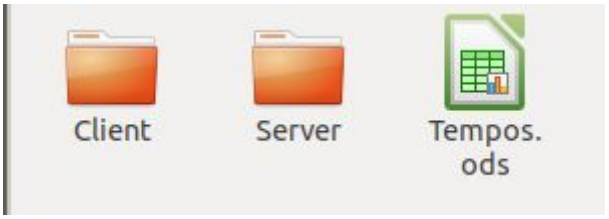

Figura 4 - Organização da pasta do projeto

Obs: Para melhor visualização da estrutura organizacional do projeto acesse https://github.com/jpsoubihe/Redes/tree/master/Projeto2

- **Desenvolvimento**

1. TCP

Primeiramente, trabalhamos no lado do servidor, preparamos todo a infraestrutura necessária para o estabelecimento de uma conexão TCP. Determinamos valores constantes PORT, MAXDATASIZE e BACKLOG representando a porta em que os clientes irão enviar as suas requisições, o máximo tamanho de um pacote de dados tanto recebido quanto enviado e o número máximo de conexões simultâneas que nosso servidor suportará.

Outra escolha feita foi o modo da coleta de dados. Optamos por recorrer ao acesso aos arquivos e captar os dados através da sua leitura. Assim, após qualquer alteração no perfil de um usuário nosso servidor continua consistente e preparado para o envio de informações. Importante ressaltar que, apesar dessa consistência, perdemos eficiência, a execução fica um pouco mais lenta devido às recorrentes manipulações nos arquivos completos. Para melhor compreensão e organização, criamos um arquivo arq.h com o

protótipo de funções que fizemos para a manipulação de arquivos, implementado em arq.c. Ficamos sempre muito atentos para o modo de abertura do arquivo, para que não ocorra sobrescrita enquanto um outro cliente requer a leitura do arquivo.

Ao começar os trabalhos na main() declaramos um vetor do tipo struct FileInfo, tal estrutura armazenará o nome do arquivo, para possibilitar o acesso posterior e um descritor do arquivo, para a leitura e eventual manipulação do conteúdo do arquivo.
Através da função getaddrinfo() setamos as informações de endereçamento do nosso servidor, como a porta e o tipo de protocolo para interação. Criamos o socket com as rotinas socket() e setsockopt() e o associamos a porta determinada através da função bind().
Com nosso ambiente preparado, ordenamos o server a “ouvir” possíveis requisições de conexão (listen()). Quando isso ocorre, a função accept() aceita o pedido, replica o processo através de um fork() e inicia a execução de nosso programa enviando uma requisição de login ao cliente, aceito com o envio de um email válido como resposta.
A partir desse momento daremos maior enfoque ao processo filho resultante do fork(), mas é importante salientar que o processo pai continua a ser executado, paralelamente, executando a função listen() e aguardando novas conexões.
É agora que a troca de informações cliente-servidor se intensificam. Foi tomado um cuidado para que as funções de envio e recebimento fossem executadas o menor número de vezes possível durante a conexão, de forma a diminuir o congestionamento da rede e, por consequência, a ocorrência de erros, mesmo considerando a simplicidade do projeto e das informações requisitadas.

Utilizamos 3 funções diferentes, mas com funções equivalentes, para o envio dos pacotes, variando de acordo com o tamanho em bytes do conteúdo a ser enviado:

1. send()
2. sendall()
3. write()

E 2 funções para a leitura:

1. recv()
2. read()

A função send() exige como parâmetros um inteiro como socket descriptor, uma referência ao endereço onde o conteúdo a ser

transmitido está armazenado, o tamanho da mensagem e as flags que indicam o tipo de transmissão (que para esse projeto setamos sempre como 0, indicando transmissão da dados “normal”).

Na implementação, sempre procuramos juntar o máximo de conteúdo possível em um único pacote, a fim de evitar o acúmulo de pacotes e aumentar o congestionamento na conexão, aumentando a chance de erros ou perda de informação. Determinamos um tamanho máximo para o pacote MAXDATASIZE, como citado acima.

A função sendall() impõe que toda a informação armazenada no buffer para envio seja enviada. Caso a informação fosse maior que o valor máximo, a função sendall() dividia a mensagem e enviava pacotes até que o conteúdo fosse entregue integralmente ao cliente.

No projeto, enviamos sempre uma string, que varia dependendo do que o cliente pede ao servidor. Tal conteúdo será armazenado pelo cliente, que salvará em arquivos de nome específico, em seu diretório, o texto, ou a imagem, entregue.

Para que o programa do cliente mantenha a execução até o momento em que ocorra um erro ou o mesmo faça uma requisição para encerrar a conexão, inserimos toda a lógica de nossas operações dentro de um loop, mantido por uma flag cuja funcionalidade é de apenas manter a execução do programa no lado do cliente.

No início de cada iteração desse loop imprimimos o menu e o cliente pode escolher que operação gostaria de executar. Essas operações serão representadas por números inteiros, que depois de convertidos para ‘long’ são enviados ao servidor. Dependendo da operação, o cliente enviará ou não novas mensagens seguindo com a interação cliente-servidor.

Para cessar a conexão, o cliente opta pela função 0. Tal função envia uma mensagem ao servidor, sinalizando o fim da comunicação, e fecha o socket dedicado à conexão. O lado servidor recebe essa mensagem e também fecha o socket que tinha reservado, abrindo espaço para outros clientes.

2. UDP

Para a implementação de uma comunicação cliente-servidor seguindo o protocolo UDP, devemos nos atentar às diferenças com o protocolo TCP.

Não é realizada uma conexão prévia entre o servidor e o cliente, como o connect() no TCP. Logo, cada pacote deve conter as informações de endereçamento, indicando seu destino. Cabe ao servidor receber o pacote, o processar e devolvê-lo ao respectivo endereço. Definimos então as portas em que o servidor e o cliente serão executados. A struct addrinfo será a responsável por armazenar as informações de endereçamento, como a porta e o endereço IP da origem do pacote. A struct sockaddr atribui ao socket um endereço onde os dados do

pacote recebido serão armazenados, tanto para o lado do cliente quanto para o lado do servidor.
Setamos as informações do socket e o associamos a constante MYPORT, que indica o número da porta que, no caso, o socket será responsável.
A função para envio de informações no protocolo UDP é a função sendto(), que tem como parâmetros o socket que enviará esses dados, a string contendo essa informação e o seu tamanho, um inteiro que representa eventuais flags para caracterizar o envio e dados sobre o endereço para onde a informação será enviada.
A função usada para recebimento de pacotes foi a rcvfrom(). Tal função também denomina um socket a ser lido, uma variável, no caso uma string, onde o conteúdo do pacote recebido será armazenado, o tamanho máximo a ser lido, eventuais flags e, assim como a função anterior, também precisa de informações sobre o endereço do computador que lhe enviou o pacote, para que numa eventual resposta a máquina saiba para onde enviar o seu pacote.
Importante salientar que, ao contrário do servidor TCP, o server UDP não replica o seu processo de maneira paralela (através do fork()), segundo a especificação do seu protocolo o servidor UDP trata cada pacote na ordem de chegada, os demais pacotes aguardam em uma fila e são tratados posteriormente, seguindo essa ordem.
Começamos a interação com o cliente enviando ao servidor um pacote contendo a conta cujas informações serão extraídas, um endereço de email.
O servidor recebe esse pacote, concatena a extensão .txt e confere se o arquivo de mesmo nome existe em seu repositório. Em caso negativo, o servidor envia um pacote contendo uma string que sinaliza ao cliente o erro de login e o processo cliente se encerra. Já caso login aceito, o servidor acessa o arquivo e, através da manipulação do mesmo, armazena em uma string o nome e sobrenome do respectivo usuário. Tais dados são enviados ao cliente, que criará um novo arquivo .txt em seu diretório para armazená-los.
Após o processo descrito anteriormente, o cliente ainda espera receber a imagem. O servidor, após envio do texto, lê o arquivo .jpeg como um arquivo binário e captura o tamanho real do arquivo e armazena o seu conteúdo em uma string. Para a correta interpretação da imagem no processo cliente, é preciso o envio do tamanho do arquivo a ser enviado antes de realmente enviar o seu conteúdo. Para que a imagem seja “montada” na aplicação cliente é necessário a leitura de caracter por caracter, assim, o tamanho real do arquivo nos serve como parâmetro para efetuarmos a leitura corretamente.
Por fim, enviamos a string com o conteúdo da imagem e a lemos, como citado acima, caracter por caracter de modo a impedir más interpretações por restrições da linguagem.

Um detalhe importante é o fato de que o cliente UDP não possui a garantia de resposta ao enviar um pacote de dados, o mesmo pode se perder, tanto no caminho cliente-servidor quanto no caminho servidor-cliente. Com isso, o uso da função select() no envio de mensagens foi extremamente importante, pois com ela pudemos estabelecer medidas para o tratamento dessa situação.

- **Análises**

Para efetuar as análises de tempo de operação, utilizamos a função gettimeofday() para medir o tempo TS de consulta no servidor e o tempo TT de consulta no cliente (Como mostrado na referência [2]). A partir destes dois tempos, determinamos o tempo de comunicação como sendo TT - TS. Medimos 20 execuções de cada operação, tanto da comunicação UDP quanto da TCP, ambas em uma mesma execução de cada servidor. Usamos dois computadores em uma mesma rede local (Rede do IC 3.5) durante a manhã, ou seja, um horário de baixo uso da rede.
Importante ressaltar que a média , o desvio padrão, o erro padrão e a margem de erro, foram calculadas pelas fórmulas descritas em (*),(**),(***),(****). Seguimos então os passos indicados por [3].

(*) média = $\sum Xi/n$, onde Xi é a amostra e n o número de amostras no total
(**) desvio padrão = $[\sum(Xi - media)^2/n]^{1/2}$
(***) erro padrão = desvio padrão / $n^{1/2}$
(****) margem de erro = erro padrão * W(Z/2), onde Z representa o nível de confiança exigido e W() mostra que o valor foi retirado de [4]

Sendo assim, sabemos que os resultados poderão oscilar de Tempo de comunicação ± margem de erro, determinada pelo nível de confiança de 95%.

Os resultados obtidos estão abaixo:

A. UDP

UDP		
SERVER(TS)(ms)	CLIENT (TT)(ms)	TT – TS (ms)
29,079	33,253	4,174
1,559	7,408	5,849
0,779	6,488	5,709
0,803	6,699	5,896
0,782	6,371	5,589
1,415	7,056	5,641
0,779	6,7	5,921
0,86	6,613	5,753
0,603	6,389	5,786
1,424	7,251	5,827
0,699	6,332	5,633
0,89	6,559	5,669
1,002	6,731	5,729
1,401	7,212	5,811
0,801	6,64	5,839
0,796	6,578	5,782
0,894	6,7	5,806
0,892	6,74	5,848
1,437	7,226	5,789
0,61	6,379	5,769
	Média	5,691
	Desvio padrão	0,358413867
	Erro padrão	0,080143777
	Margem de erro	0,157081803

Tabela 1 - Tempos de consulta e comunicação UDP (em ms)

B. TCP

TCP		
SERVER(TS)	CLIENT (TT)	TT – TS (ms)
1,046	4,409	3,363
0,512	3,346	2,834
0,693	3,565	2,872
0,634	3,433	2,799
0,616	3,826	3,21
0,604	3,731	3,127
0,638	3,372	2,734
0,523	3,537	3,014
0,96	3,737	2,777
0,681	3,524	2,843
0,465	3,582	3,117
0,603	3,574	2,971
0,487	3,846	3,359
0,604	3,797	3,193
0,622	3,512	2,89
0,604	3,481	2,877
0,567	3,487	2,92
1,02	3,711	2,691
0,615	3,55	2,935
0,578	3,327	2,749
	Média	2,96375
	Desvio padrão	0,196517652
	Erro padrão	0,043942683
	Margem de erro	0,086127658

Tabela 2 - Tempos de consulta e comunicação TCP (em ms)

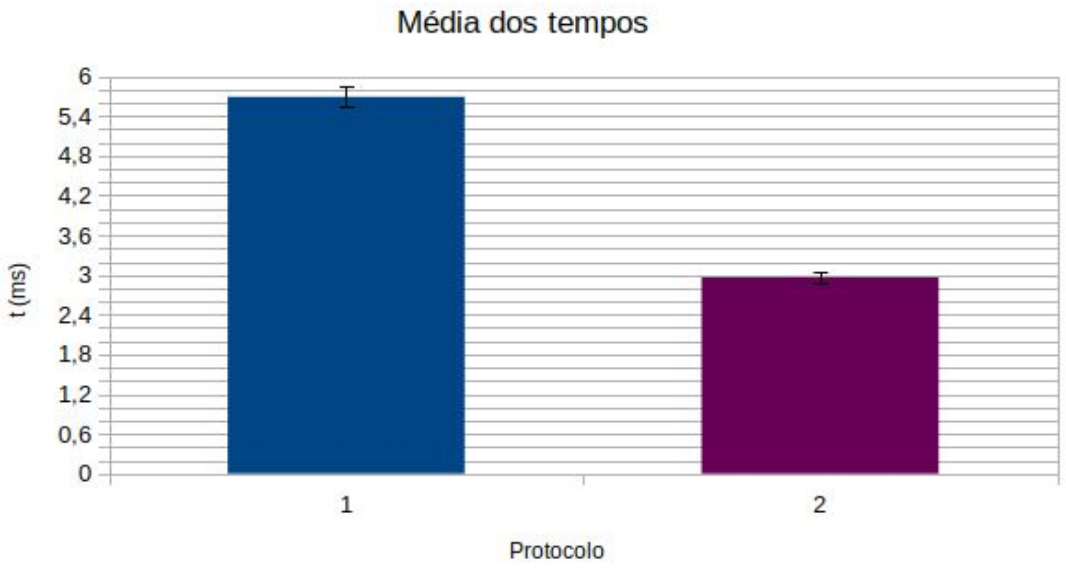

Gráfico 1 - Médias de tempo coletados por protocolo (em ms)
obs: 1 = UDP; 2 = TCP

Por essas análises é interessante notar que, ao contrário do que esperávamos, os tempos de execução da operação pré determinada por [5] por uma comunicação TCP foram menores que os medidos na comunicação UDP, que como característica tem a rapidez de sua transmissão.
Também vemos que as medições de tempo foram bem precisas levando em conta a margem de erro retratada no *Gráfico 1*. Assim, mesmo com o resultado inesperado podemos considerá-lo assertivo.

- ## Conclusão

  Com a conclusão do projeto, pudemos adquirir uma visão crítica sobre os dois protocolos e tirar certas conclusões sobre suas características. Devemos destacar algumas semelhanças em sua implementação, o baixo nível da linguagem C nos permite ver isso, e o resultado surpreendente que nos sucedeu com relação à medição do tempo de comunicação. Segundo a teoria, o protocolo UDP seria responsável por uma transmissão mais rápida, por não precisar estabelecer uma conexão prévia, mas em nossa análise a execução da funcionalidade pelo protocolo UDP apresentou uma média de tempos maior que a presente com o protocolo TCP. Justificamos esse fato por uma possível sobrecarga de trabalho do SO ao quebrar o grande volume de mensagens em vários pacotes, por causa do datagrama de tamanho limitado no protocolo UDP, ao contrário da stream de dados presente no TCP. O fato dos computadores estarem muito próximos um do outro, na rede local do IC pode nos fornecer uma visão enganosa, pois dados relevantes como tráfego e atrasos relativos a transmissão pouco influenciam em nossas medições.
  Quanto ao tamanho de código da nossa implementação, os dois protocolos foram relativamente semelhantes. Foram por volta de 60 linhas a mais para a implementação do servidor TCP comparado ao servidor UDP, pois nessa execução foi implementada a funcionalidade de login, o que justifica essa diferença. Já no lado cliente o cliente TCP apresentou pequena diferença em frente ao UDP. Podemos considerar, neste caso, que o tratamento de perdas de pacotes no UDP balanceou a diferença, mesmo com o login do TCP.
  Quanto a erros, o protocolo TCP apresentou mais restrições no envio de imagens .JPEG, mas, por outro lado, podemos considerar como certeza a chegada da informação no outro lado da comunicação, ao contrário do protocolo UDP, no qual tratamos, sempre que enviamos uma mensagem, o caso em que o pacote de informações se perde durante sua transmissão, através da função select().

- ## Referências

[1] http://beej.us/guide/bgnet/

[2] www.ic.unicamp.br/~edmundo/MC833/mc833/computar_tempo_trab.pdf

[3] https://pt.wikihow.com/Calcular-o-Intervalo-de-Confian%C3%A7a

[4] https://www.statisticshowto.datasciencecentral.com/tables/z-table/
[5] www.ic.unicamp.br/~edmundo/MC833/turma1s2019/proj2_18331s19.pdf