Nome: André Costa Werneck
Matrícula: 2017088140
Data: 04/09/2020
Disciplina: Redes de Computadores

# DOCUMENTAÇÃO DO TRABALHO PRÁTICO 1

## Introdução

Este documento tem como objetivo explicar sucintamente as escolhas tomadas na implementação do TP1.

Pediu-se que fosse implementado, em linguagem C ou C++, um jogo da forca em um programa híbrido, fazendo uso do protocolo TCP com implementações tanto para IPv4 quanto para IPv6. Sugeriu-se fazer uso da biblioteca “socket.h”.

## Objetivos

O objetivo do trabalho é o de proporcionar aos alunos uma experiência prática do que foi visto em sala de aula. Visava-se, com isso, o aprimoramento da teoria de redes de computadores, principalmente o entendimento de como funcionam as comunicações entre cliente e servidor, o uso do protocolo TCP e, também, o desenvolvimento de habilidades de programação em geral.

## Implementação

Primeiramente, escolheu-se implementar o trabalho na linguagem C++, uma vez que ela é orientada à objetos e possui importantes estruturas de dados já implementadas, com a classe String, que são muito úteis.

### SERVIDOR

Decidiu-se começar a implementação com o servidor.

Para montar o esqueleto do servidor, usou-se as vídeo aulas do Professor Ítalo como base.

Logo de cara, resolveu-se fazer a seguinte função:

```
void usageServer(char **argv) {
   cout << "Usage:" << argv[0] << " <server port> " << endl;
   cout << "example: " << argv[0] << " 5151 " << endl;
   exit(EXIT_FAILURE);
}
```

Ela tem como objetivo pegar os argumentos passados na linha de execução e ensinar ao usuário como executar corretamente o programa.

Após ela, criou-se a função main() que começou com o seguinte trecho:

```
int main(int argc, char **argv) {
    if (argc != 2) {
        usageServer(argv);
    }
```

Nele, conferiu-se a quantidade de argumentos passados à linha de comando que, de acordo com a especificação passada, deveria ser exatamente 2 (./servidor <porta>). Caso os parâmetros fossem passados incorretamente, a função usageServer era chamada.

No próximo trecho, declarou-se variáveis relacionadas à conexão e inicializou-se-as. Rapidamente, viu-se a necessidade de implementar uma função de parsing para estudo do endereço a ser utilizado para o servidor, uma vez que o programa deveria ser híbrido. A ela foi dado o seguinte cabeçalho:

```
int onServerParseAddress(string protocol, const char
*portStr,struct sockaddr_storage *storage)
```

Ela foi implementada em um programa separado, chamado de **common**. Decidiu-se criar o arquivo common.cpp para que fossem implementadas funções de uso importante e comum entre servidor e cliente. Nesse caso, a função era útil apenas para o servidor, mas foi escolhido que ficaria no arquivo common por facilidade e menor poluição no arquivo principal.

Basicamente, ela recebe o protocolo a ser utilizado(IPv4 ou IPv6), a porta e a estrutura sockaddr_storage. O protocolo foi passado via string e foi definido como uma constante na parte de cima do arquivo servidor.cpp, como aconselhado pelos professores. A porta era lida da linha de comando e a estrutura, é padrão da biblioteca socket e serve para guardarmos a família do protocolo e depois apontar para uma estrutura do tipo IPv4 ou IPv6. Logo, ela é muito útil para quem deve fazer um programa que rode em ambos os protocolos. Fez-se de um jeito que o servidor poderia se conectar a qualquer porta disponível localmente.

Após isso, fez-se uso da função setsockopt(), também padrão da socket.h visando reutilizar uma porta antes usada, para que o programa não fique travado e as portas indisponíveis por certo tempo após primeiro uso. Logo depois, inicializou-se o endereço do servidor e começou-se o processo da conexão, tendo em vista a sequência:

- bind()
- listen()
- accept()

Com a conexão aceita, declarou-se um loop while(true), para que criássemos uma espécie de chat entre cliente e servidor, até que um dos dois encerrasse a conexão com o outro.

Para dar início ao jogo da forca, a palavra, que deveria ser pré-definida, foi definida constante em forma de uma string no topo do arquivo e seu conteúdo era “trabalho”. Com isso feito, mandou-se a mensagem de início de jogo,contendo o uint8_t 1 e o uint8_t tamanho da palavra, exatamente como especificado. Após isso, fez-se uma sequência baseada na seguinte ordem:

- conferir se a palavra já tinha sido adivinhada
- se não:
- receber os palpites do cliente
- análise do tipo da resposta e, se correto, cálculo da frequência e posições da respectiva letra palpitada.
- Escrever e enviar a resposta ao cliente.
- se sim:
- enviar mensagem contendo uint8_t 4, indicando o fim do jogo
- encerrar a conexão.

**Cliente**

Implementou-se uma função usageClient, com os mesmos objetivos da criada para o servidor. Ela segue:

```
void usageClient(char **argv) {
    cout << "Usage:" << argv[0] << " <server IP> <server port> " << endl;
    cout << "example: " << argv[0] << " ::1 5151 " << endl;
    exit(EXIT_FAILURE);
}
```

Além disso, criou-se uma estrutura Message, que recebia tipo, tamanho e caractere. Ela foi criada visando padronizar e facilitar o armazenamento de todos os detalhes relativos às mensagens enviadas e recebidas e a manipulação delas. Ela foi declarada assim:

```
struct Message {
    uint8_t type; // 8 - bits for the type
    uint8_t size; // 8 - bits for the size of the word
    char character; // 8 - bits for the character
};
```

Após isso, de maneira análoga ao que foi feito no servidor, verificou-se os parâmetros passados via linha de comando ao cliente e, caso estivessem errados, ensinava-se ao usuário como executar o arquivo corretamente. Também de forma semelhante, para fazer o parsing do enderço e protocolo a ser utilizado, criou-se a seguinte função:

```
int onClientParseAddress(const char *addressStr,const char
*portStr, struct sockaddr_storage *storage)
```

Ela recebe o endereço passado e a porta do servidor, ambos passados via linha de comando para o cliente. Além disso também possui um ponteiro para a estrutura sockaddr_storage, que já foi explicada acima. Seu funcionamento é idêntico ao da função onServerParseAddress que também já foi explicada neste documento.

Dessa forma, inicializou-se o endereço e socket corretamente. Após isso, sequência para a conexão foi:

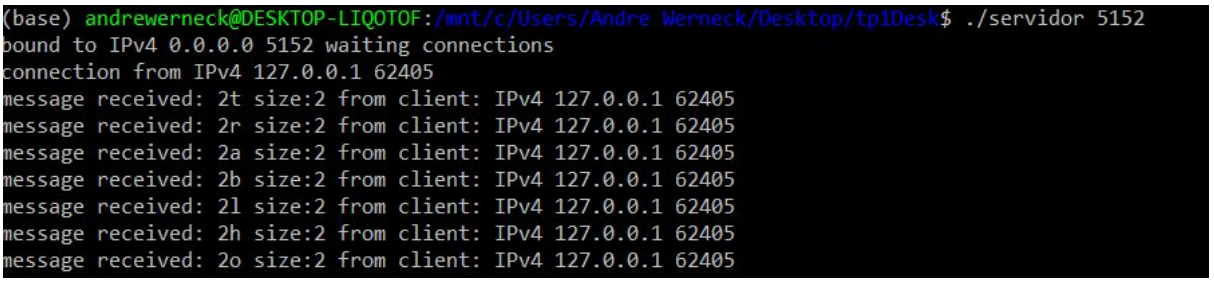
- connect()

Depois, ficou-se esperando a mensagem de início de jogo a ser mandada pelo servidor. Assim que recebida, fez-se um loop do tipo while(true) para que fosse efetivado o chat entre cliente e servidor. O algoritmo após o while foi:

- se tipo da primeira mensagem == 1:
- começa o jogo
- verifica-se se recebeu-se a mensagem de fim de jogo
- se não:
- palpita-se a primeira letra e envia-a ao servidor
- recebe a resposta
- processa a resposta a saber se acertou ou errou
- tenta novamente
- se sim:
- imprime na tela que acabou o jogo e encerra-se a conexão com o servidor.

São exemplos do chat via socket entre cliente e servidor as imagens abaixo:

servidor



cliente

```
(base) andrewerneck@DESKTOP-LIQOTOF:/mnt/c/Users/Andre Werneck/Desktop/tp1Desk$ ./cliente 127.0.0.1 5152
connected to IPv4 127.0.0.1 5152
Type = 1 and Size = 8
Game Started
try letter = t
Answer from server= 310
try letter = r
Answer from server= 311
try letter = a
Answer from server= 3224
try letter = b
Answer from server= 313
try letter = l
Answer from server= 315
try letter = h
Answer from server= 316
try letter = o
Answer from server= 317
Word was discovered and the game has ended!: Success
```

As mensagens acima provam que o projeto servidor-cliente está em pleno funcionamento.

**Common**

Além das funções já explicitadas acima, foram implementadas as seguintes funções no arquivo de uso comum:

```
void logexit(const char *message);
```

Padroniza a saída do programa e indica uma mensagem de erro ao usuário. Busca facilitar o entendimento do mesmo acerca do ocorrido.

```
int buf_ascii2int(signed char c);
```

Converte char lido do buffer para inteiro de acordo com a tabela ascii. Muito útil para agilizar a comparação das mensagens recebidas tanto do cliente quanto do servidor.

```
void addressToString(struct sockaddr *address,char *str,size_t
sizeOfStr);
```

Sugerida pelo Professor Ítalo, serve para converter o endereço para string e facilitar sua apresentação ao usuário.

**Considerações importantes**

- Para todas as mensagens passadas, foram utilizados os tipos uint8_t, para inteiros e char, para caracteres. Ambos possuem tamanho definido de 8 bits, ou seja, 1 byte. Dessa forma, tanto a forma, quanto o tamanho das mensagens enviadas respeitou as exigências passadas pelos professores.
- Definiu-se o tamanho do buffer como 256 que equivale a 2^8.
- O snprintf acrescenta um ‘/0’ ao final de todas as strings escritas no buffer.Isso é padrão da linguagem e este último byte representa o final da string. Dessa forma, foi necessário passar ‘tamanho da mensagem’ +1.

- Fez-se uso do namespace std.
- Todas as mensagens impressas no terminal visam facilitar o entendimento do usuário.

## Conclusão

O trabalho prático sugerido foi muito interessante e de extrema valia, uma vez que ficaram muito mais claros os conceitos aprendidos nas aulas teóricas e foram aprimoradas as habilidades de programação em geral, já que enfrentou-se dificuldades para implementar o TP. Conclui-se, portanto, que trabalhos como este são de suma importância na formação do aluno.