

# Documentação Trabalho Prático 1

DCC023 - Redes de Computadores  
UNIVERSIDADE FEDERAL DE MINAS GERAIS

Frederico Ribeiro Queiroz

31 de agosto de 2020

## 1 Introdução

O objetivo deste trabalho é implementar um jogo da forca simplificado, jogado entre cliente um e um servidor, que se comunicam através de *aplicações sockets*.

Em um jogo da forca, o jogador deve dar palpites até acertar qual a palavra proposta, sabendo a quantidade de letras presente na mesma. Será utilizada uma versão simplificada onde os palpites errados não são contabilizados (o jogador não é ‘enforcado’) e o jogo acaba quando todas as letras da palavra são encontradas.

Neste jogo, o cliente envia letras como palpites e o servidor recebe e responde os locais de ocorrência da letra, dada que ela exista na palavra. Os detalhes serão discutidos nas seções seguintes.

## 2 Soluções implementadas

Os programas foram implementados em *linguagem C*. Para realizar a comunicação entre cliente e servidor, foram implementadas aplicações sockets que utilizam o protocolo TCP. O cliente e o servidor implementados suportam simultaneamente tanto endereços IPv4 quanto IPv6, sem necessidade de alteração em código fonte. O detalhe sobre essa interoperabilidade de versão será discutido em uma subseção posterior.

### 2.1 Protocolo de comunicação

O protocolo entre cliente e servidor possui quatro tipo de mensagens, da seguinte forma:

Mensagem	Descrição	Conteúdo		
Mensagem 1	Início de jogo	tipo (1 byte)	tamanho da palavra (1 byte)	
Mensagem 2	Palpite	tipo (1 byte)	caracter a ser testado (1 byte)	
Mensagem 3	Resposta	tipo (1 byte)	número de ocorrências (n) (1 byte)	posições (1 byte * n)
Mensagem 4	Fim de jogo	tipo (1 byte)		

Por facilidade de implementação, foi criado apenas uma estrutura (*struct MessageInfo*) que contém todos os campos utilizados pelas mensagens.

Estrutura criada:

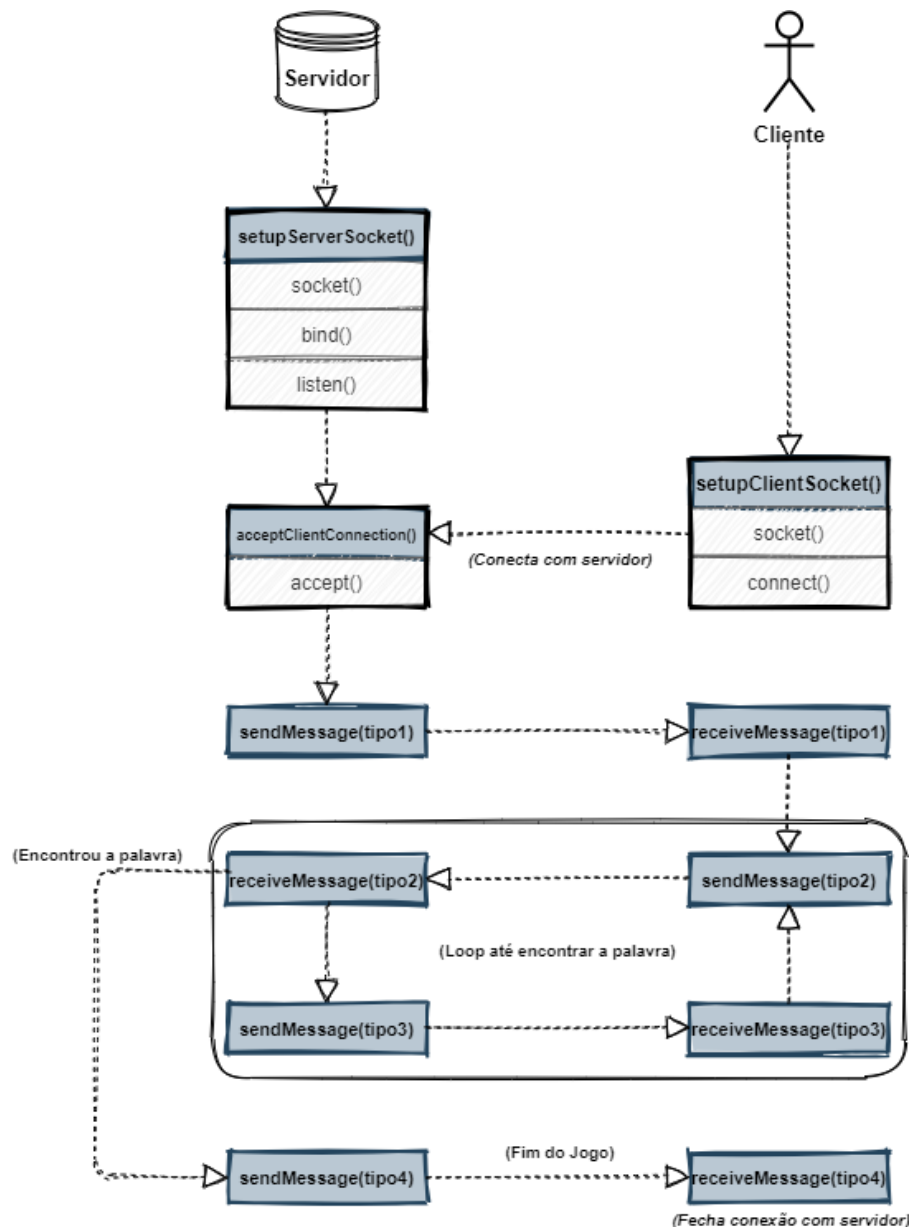
```
typedef struct MessageInfo {  
    u_int8_t messageType; // Used in Messages {1,2,3,4}  
    u_int8_t wordSize; // Used in Message {1}  
    u_int8_t guessedLetter; // Used in Message {2}  
    u_int8_t occurrencesNumber; // Used in Message {3}  
    u_int8_t occurrencesPosition[MAX_OCCURRENCES]; // Used in Message {3}  
} Message;
```

**Figura 1:** Estrutura criada para armazenar mensagens

Dessa forma, foi possível utilizar uma única função *sendMessage()* para enviar e uma *recieveMessage()* para receber mensagens, tanto para o cliente quanto para o servidor. As duas funções e a estrutura da mensagem se encontram na biblioteca comum *src/lib/-protocolUtility.h*.

## 2.2 Funcionamento básico dos programas

O esquema a seguir apresenta visualmente o funcionamento básico e o relacionamento dos dois programas:



**Figura 2:** Funcionamento básico do Servidor e Cliente

O servidor utiliza a função `setupServerSocket()` para estabelecer um socket. O cliente utiliza a função `setupClientSocket()` para estabelecer um socket, e em seguida, se conectar ao servidor. A conexão é aceita pelo servidor através da função `acceptClientConnection()`. A partir desse ponto, servidor e cliente começam a trocar mensagens através das funções `sendMessage()` e `receiveMessage()` até que o cliente acerta a palavra e receba a mensagem de fim de jogo.

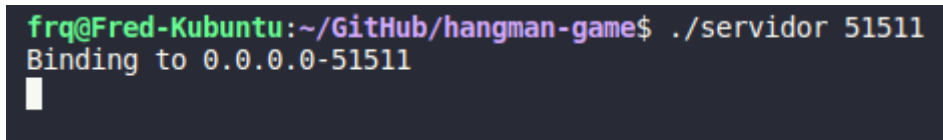
## 2.3 Interoperabilidade de versões (IPv4 e IPv6)

Graças a função `getaddrinfo()` da biblioteca `sys/socket.h` (link para o manual nas referências) o cliente e o servidor são independentes de versão de IP, inclusive podendo se conectar usando versões distintas (servidor IPv4 e cliente IPv6, por exemplo).

A ideia geral é definir argumentos para a `getaddrinfo()` que façam com que ela retorne ambos os endereços (IPv4 e IPv6) e utilize o primeiro endereço que funcione. Isso é possível graças a uma classe de endereçamento que mapeia “v4-para-v6”. Os detalhes de implementação dessa função vão além do escopo deste trabalho, portanto serão omitidos.

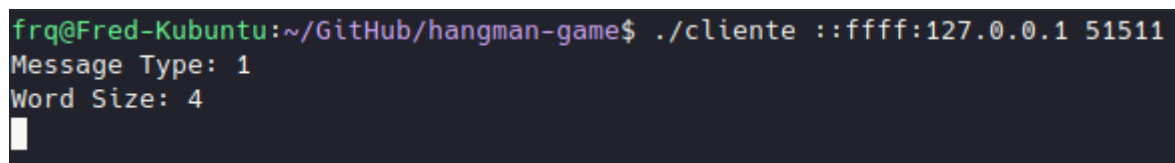
A seguir um exemplo de interoperabilidade entre o servidor e o cliente.

Neste caso, o servidor está conectado a um endereço IPv4 e solicitaremos a conexão de um cliente passando um endereço IPv6.



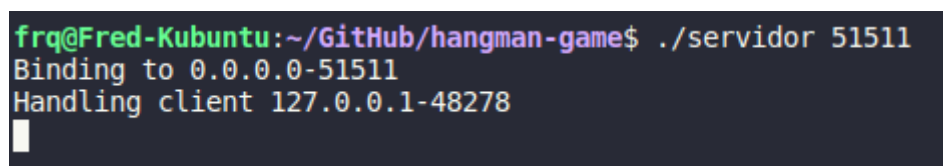
```
frq@Fred-Kubuntu:~/GitHub/hangman-game$ ./servidor 51511
Binding to 0.0.0.0-51511
█
```

**Figura 3:** Servidor conectado a um socket IPv4



```
frq@Fred-Kubuntu:~/GitHub/hangman-game$ ./cliente ::ffff:127.0.0.1 51511
Message Type: 1
Word Size: 4
█
```

**Figura 4:** Cliente conectando com endereço IPv6 se conectando



```
frq@Fred-Kubuntu:~/GitHub/hangman-game$ ./servidor 51511
Binding to 0.0.0.0-51511
Handling client 127.0.0.1-48278
█
```

**Figura 5:** Servidor recebendo conexão do cliente com endereço IPv4

O que está acontecendo aqui é que o servidor está ‘ouvindo’ um socket IPv4, e o cliente está tentando conectar utilizando um socket IPv6. Neste ponto em que o cliente tenta se conectar, o socket ainda não está limitado a um endereço específico, então a implementação do lado do cliente irá reconhecer que está se conectando a um servidor com endereço IPv4 e atribuir um endereço v4 mapeado a partir do endereço v6 ao socket no momento de executar a função `connect()`.

### 3 Estrutura básica do projeto