UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ARTES, CIÊNCIAS E HUMANIDADES

ACH2026 – Redes de Computadores
2^o semestre de 2025

Exercício Programa

Prof. Dr. Renan Cerqueira Afonso Alves

Sumário

1 Introdução

O exercício programa consistirá na implementação de um protocolo de camada de aplicação simples para suportar o desenvolvimento de um jogo da forca online. Além disso, a interface de sockets será utilizada para desenvolver as aplicações que usam o protocolo.

1.1 Regras do jogo da forca

Nesta seção, vamos definir como é o funcionamento do jogo da forca que adotaremos para este exercício.

No jogo da forca, um jogador mestre propõe uma palavra que deve ser adivinhada pelos outros jogadores. Inicialmente, o mestre revela apenas a quantidade de letras da palavra.

Os outros jogadores alternam-se dando palpites sobre as letras que acham que a palavra contém. Se o palpite for correto, o jogador mestre revela o posicionamento da letra na palavra. Caso contrário, os outros jogadores perdem uma vida. Alternativamente, um jogador pode tentar acertar a palavra inteira em vez de sugerir apenas uma letra.

O jogo termina quando todas as letras da palavra forem reveladas, algum jogador acertar a palavra ou se as vidas acabarem.

Veja um exemplo com três jogadores: Alice, Bob e Chris.

Alice é escolhida como mestre e pensa na palavra "BANANA". Ela revela os outros que a palavra tem 6 letras:

```
 -----
 |   |
 |
 |
 |
 |
------   _ _ _ _ _ _
```

Em seguida, Bob dá o palpite da letra "A":

```
 -----
 |   |
 |
 |
 |
 |
------   _ A _ A _ A
```

Em seguida, Chris dá o palpite da letra "R":

```
 -----
 |   |
 |   o
 |
 |
 |
------   _ A _ A _ A
Palpites errados: r
```

O próximo palpite de Bob é "B":

```
 -----
 |   |
 |   o
 |
 |
 |
------   B A _ A _ A
Palpites errados: r
```

Na vez de Chris, ele diz que acha que a palavra é "BATATA", mas, como ele errou, uma vida é perdida.

```
 -----
 |   |
 |   o
 |   l
 |
 |
------   B A _ A _ A
Palpites errados: r, batata
```

Na próxima jogada, Bob diz que acha que a palavra é "BANANA"e vence o jogo.

Adotaremos que, no começo do jogo, os jogadores possuem 7 vidas, ou seja, o máximo de erros que podem ser cometidos pelos jogadores é igual a 6. Ao cometer o sétimo erro, o jogo termina em derrota. No desenho abaixo, vemos a forca com 6 erros cometidos (cada erro adiciona uma parte do corpo no desenho: cabeça, tronco, braço esquerdo, braço direito, perna esquerda e perna direita).

```
 -----
 |   |
 |   o
 | --l--
 |  / \
 |
------
```

2 Arquitetura

Adotaremos uma arquitetura cliente-servidor, na qual a aplicação servidor é responsável somente por coordenar o jogo (ou seja, não é possível faz palpites através da aplicação servidor). Por outro lado, a aplicação cliente representa os jogadores, permitindo que façam palpites. Uma das funções do servidor é escolher quais dos jogadores é o mestre da rodada (o mestre é o jogador que escolhe a palavra a ser adivinhada).

3 O protocolo Hangman

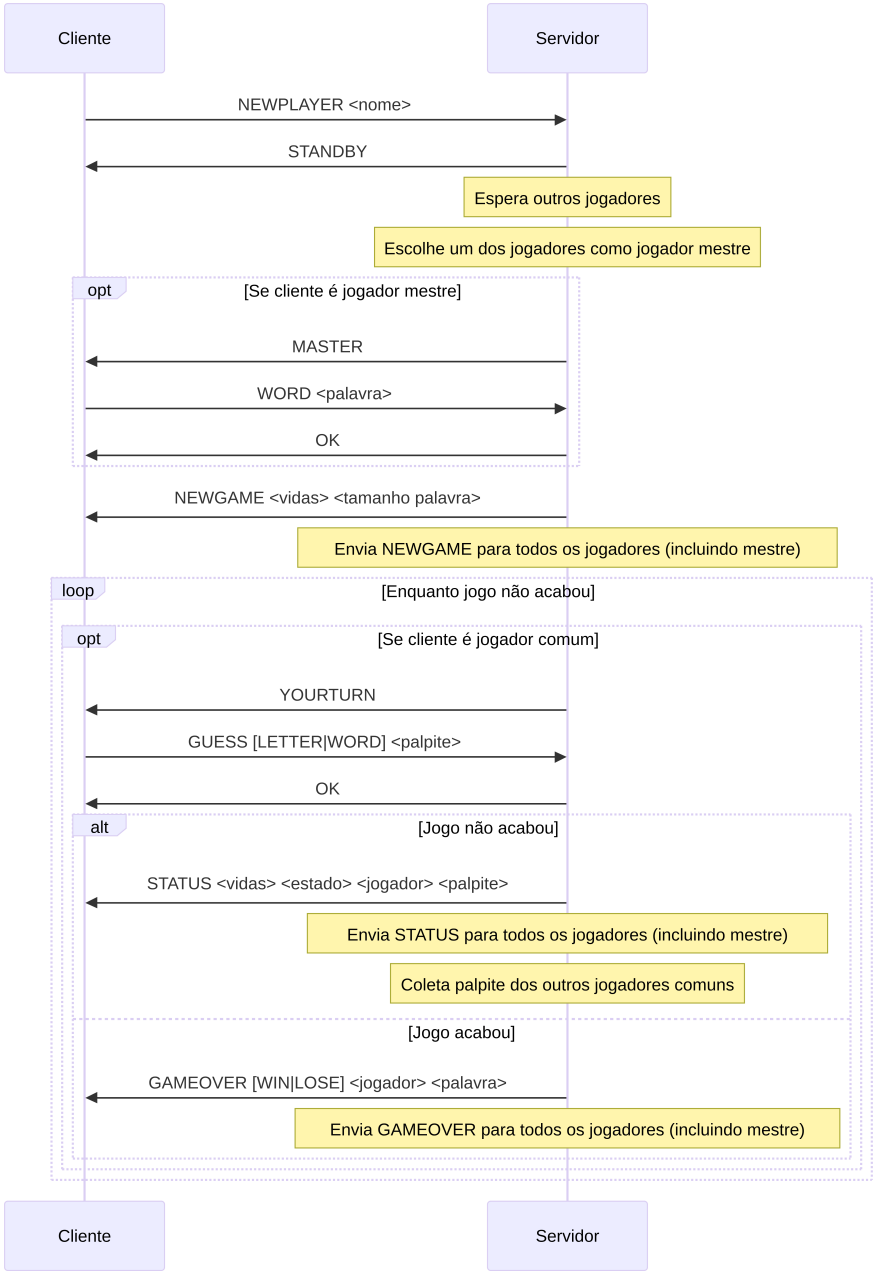

Figura 1: Protocolo Hangman: fluxo normal

A Figura 1 resume o funcionamento e as principais mensagens do protocolo Hangman, que deve ser implementado pelas aplicações cliente e servidor. O protocolo é **stateful**, ou seja, o servidor deve armazenar informações referentes ao estado atual do jogo e dos clientes conectados. Além disso, a troca de mensagens é realizada via conexões TCP **persistentes**.

O fluxo de execução do protocolo inicia-se com os clientes enviando uma mensagem `NEWPLAYER <nome>` para o servidor, onde <nome> deve ser substituído pelo nome do jogador. O servidor responde com uma mensagem `STANDBY`, indicando para o cliente aguardar a próxima mensagem do servidor.

Após todos os jogadores se conectarem (a quantidade exata de jogadores depende da implementação), o servidor deve escolher um deles para ser o jogador mestre. Apenas para o jogador mestre, o cliente envia uma mensagem `MASTER`, para a qual o cliente deve responder `WORD <palavra>`, onde <palavra> deve ser substituído pela palavra escolhida pelo jogador. O servidor, por sua vez, confirma o recebimento e processamento bem sucedido da palavra com uma mensagem `OK`.

Em seguida, o servidor dá início ao jogo enviando uma mensagem `NEWGAME <vidas> <tamanho palavra>` para cada um dos clientes (incluindo o mestre). Nesta mensagem <vidas> é um número inteiro que representa a quantidade inicial de vidas e <tamanho palavra> é um número inteiro que representa o tamanho da palavra.

A partir deste ponto, o servidor pergunta palpites aos **jogadores comuns** (um de cada vez, de forma cíclica) com uma mensagem `YOURTURN`. O jogador da vez deve responder com uma mensagem `GUESS [LETTER|WORD] <palpite>`. Há dois tipos de palpite: no tipo LETTER o palpite deve conter apenas uma letra, enquanto que no tipo WORD o jogador tenta adivinhar a palavra inteira. O servidor responde com `OK` para confirmar o recebimento e processamento correto do palpite.

Após receber um palpite e antes de perguntar o palpite do próximo jogador, o servidor verifica se o jogo acabou.

No caso do jogo não ter acabado, o servidor informa o estado atual do jogo para todos os jogadores com uma mensagem `STATUS <vidas> <estado> <jogador> <palpite>`, na qual <vidas> é um número inteiro representando a quantidade de vidas restantes, <estado> representa o estado atual do jogo, <jogador> é o nome do jogador que fez o último palpite e <palpite> é o palpite feito pelo jogador. O estado do jogo é codificado da seguinte forma: cada letra já adivinhada da palavra é revelada, enquanto que cada letra que ainda falta ser adivinhada é substituída pelo caractere `_` (underline). Por exemplo, se a palavra for `banana` e o primeiro palpite feito pelo jogador `Renan` foi `a`, a mensagem de status seria a seguinte: `STATUS 7 _a_a_a Renan a`

Se o jogo acabou, o servidor deve informar a todos os jogadores (incluindo o mestre) com uma mensagem `GAMEOVER [WIN|LOSE] <jogador> <palavra>`, indicando se a palavra foi adivinhada (WIN) ou não (LOSE). O campo <jogador> contém o nome do jogador que fez o último palpite, e o campo <palavra> contém a palavra que foi (ou deveria ter sido) adivinhada. Após enviar a mensagem `GAMEOVER`, o servidor encerra a conexão com todos os jogadores, encerrando o fluxo de execução com os clientes.

3.1 Fluxos de erro

A Figura 1 apresenta o comportamento do protocolo na ausência de erros. Esta seção descreve os possíveis erros que podem ocorrer no fluxo de execução normal. A não ser que seja explicitado, a conexão com o cliente ou servidor deve ser encerrada após enviar a

mensagem de erro.

Lista de mensagens de erro:

- `ERROR INVALID_FORMAT`: mensagem de erro enviada sempre que houver erro de formatação da mensagem recebida (ver Seção 3.2).
- `ERROR INVALID_MASTER_MESSAGE`: mensagem enviada para todos os jogadores caso o jogador mestre não envie uma palavra válida (não respondeu com mensagem `WORD`, ou a palavra estava ausente, ou a palavra contém caracteres inválidos (veja Seção 3.2).
- `ERROR UNEXPECTED_MESSAGE`: mensagem de erro enviada sempre que a mensagem recebida não for uma mensagem de erro, mas não for um dos tipos de mensagem esperada pelo protocolo naquele ponto de execução.
- `ERROR INVALID_PLAYER_NAME`: mensagem de erro enviada pelo servidor ao cliente se o nome fornecido pela mensagem `NEWPLAYER` for inválido. O nome é considerado inválido se estiver vazio, se contiver espaços, ou se contiver caracteres não alfanuméricos.
- `ERROR NOT_ENOUGH_PLAYERS`: Mensagem de erro enviada pelo servidor ao jogador mestre caso não haja jogadores comuns restantes para continuar o jogo.
- `ERROR ALREADY_GUESSED`: Mensagem de erro enviada pelo servidor se a mensagem `GUESS` do cliente contiver um palpite já enviado anteriormente (por qualquer jogador). A conexão com entre cliente e servidor **não é encerrada**.
- `ERROR INVALID_LETTER`: Mensagem de erro enviada pelo servidor se a mensagem `GUESS LETTER` do cliente contiver um caractere inválido. A conexão com entre cliente e servidor **não é encerrada**.
- `ERROR INVALID_WORD_LENGTH`: Mensagem de erro enviada pelo servidor se a mensagem `GUESS WORD` do cliente contiver uma palavra tamanho diferente da palavra a ser adivinhada. A conexão com entre cliente e servidor **não é encerrada**.
- `QUIT`: Esta mensagem não é exatamente um erro, mas pode ser enviada pelo cliente ao servidor para indicar que o jogador deseja se desconectar. O servidor deve responder com `OK` e encerrar a conexão.

3.2 Formato da mensagem

As mensagens devem seguir o seguinte formato:

- Todas as mensagens são em texto ASCII puro.
- Todas as mensagens devem ser terminadas por um caractere de carriage return (`"\r"`) seguido por um caractere de line feed (`"\n"`).
- Os campos das mensagens devem ser separados por um único caractere de espaço.
- Assuma que o nome dos jogadores enviados na mensagem `NEWPLAYER` não possui espaço, mas pode possuir qualquer letra ou número.
- Assuma que a palavra a ser adivinhada não contém hífen e é composta apenas por letras sem acento codificadas com caracteres ASCII (letras de a até z maiúsculas ou minúsculas).

4 Aplicação servidor

A aplicação servidor deve receber dois parâmetros de execução: a quantidade de jogadores suportados (obrigatório) e a porta em que o servidor irá aguardar conexões dos clientes (opcional, com valor padrão 6891).

Ao iniciar, o servidor deve escutar no endereço IP 0.0.0.0[1] na porta TCP adequada e imprimir `Servidor inicializado na porta <porta>` na saída padrão.

Em seguida, o servidor deve executar em um loop infinito. Em cada interação do loop, deve ocorrer uma execução do protocolo, conforme indicado na Seção 3, considerando a quantidade de jogadores escolhida na linha de comando e a quantidade inicial de vidas igual a 7 (hardcoded).

Ao considerar as palavras e palpites, o servidor não faz distinção entre letras maiúsculas e minúsculas.

O servidor deve imprimir o andamento do jogo na saída padrão. Ao iniciar um novo jogo, deve imprimir `Iniciando novo jogo...`, e, para cada jogador, deve imprimir `Aguardando jogador <i>... Jogador conectado: <nome>`, com i sendo um número inteiro crescente, iniciado em 1.

Em seguida, o servidor deve escolher aleatoriamente um jogador para ser o mestre, exibindo esta informação, bem como a palavra escolhida pelo mestre: `Jogador mestre: <nome>. Jogador mestre forneceu a palavra: <palavra>`. Se tudo correr bem, o jogo deve ser iniciado, imprimindo `Jogo iniciado com sucesso!`.

A partir deste ponto, os jogadores comuns alternam-se dando palpites. O servidor deve mostrar isso no seguinte formato:

```
Vez do jogador <nome>.
Processando palpite de <letra ou palavra>: <palpite>.
Continuando jogo. Estado atual: <estado>, vidas restantes: <valor>.
```

Eventualmente o jogo terminará, e o servidor deve imprimir `Jogador <nome> adivinhou a palavra!` ou `A palavra não foi adivinhada. Último jogador: <nome>`, seguido de `Finalizando jogo...`. Neste momento, as conexões com os todos clientes devem ser **encerradas**. Em seguida, o servidor fica pronto para iniciar um novo jogo.

Se ocorrer algum erro de execução, conforme explicado na Seção 3.1, basta imprimir que o erro ocorreu na saída padrão e seguir a execução.

Veja um exemplo de saída do servidor abaixo:

```
$ ./hangman-server 2 9999
Servidor inicializado na porta 9999
Iniciando novo jogo...
Aguardando jogador 1...
Jogador conectado: Glenn

Aguardando jogador 2...
Jogador conectado: Magus

Jogador mestre: Magus
Jogador mestre forneceu a palavra: lava
```

[1] Usar este endereço faz com que o servidor escute em todas as interfaces disponíveis.

```
Jogo iniciado com sucesso!

Vez do jogador Glenn.
Processando palpite de letra: 'a'
Continuando jogo. Estado atual: _a_a, vidas restantes: 7

Vez do jogador Glenn.
Processando palpite de letra: 'b'
Continuando jogo. Estado atual: _a_a, vidas restantes: 6

Vez do jogador Glenn.
Processando palpite de letra: 'l'
Continuando jogo. Estado atual: la_a, vidas restantes: 6

Vez do jogador Glenn.
Processando palpite de letra: 'p'
Continuando jogo. Estado atual: la_a, vidas restantes: 5

Vez do jogador Glenn.
Processando palpite de letra: 'v'
Jogador Glenn adivinhou a palavra!
Finalizando jogo...


Iniciando novo jogo...
Aguardando jogador 1...
(...)
```

5 Aplicação cliente

A aplicação cliente deve receber dois parâmetros de execução: o nome do jogador (obrigatório) e o endereço do servidor no formato <IP>:<PORTA> (opcional, com valor padrão 127.0.0.1:6891).

A aplicação conecta-se ao servidor utilizando o nome do jogador passado como parâmetro, seguindo o funcionamento do protocolo. As seguintes mensagens devem ser exibidas na saída padrão `Conectando ao servidor... Aguardando o jogo começar...`.

Se o cliente receber a mensagem `MASTER`, a aplicação deve pedir ao cliente que digite a palavra para os outros jogadores adivinharem: `Você é o mestre do jogo! Digite a palavra:`.

Ao receber a mensagem `NEWGAME` do servidor, a aplicação deve exibir `Jogo iniciado! Vidas para adivinhar: 7 Tamanho da palavra: <n> letras.`.

Sempre que o cliente receber uma mensagem de `STATUS`, a aplicação deve exibir o seguinte texto `Letras erradas: <lista de letras> Palavras erradas: <lista de palavras> Jogador <nome> fez uma jogada: <palpite>. Restam <n> vidas.`. Note que é responsabilidade da aplicação construir e manter as listas de palpites incorretos. **Opcional**: mostrar uma arte ASCII da forca de acordo com o número de vidas restantes.

Sempre que o cliente receber uma mensagem `YOURTURN`, a aplicação deve pedir para o usuário digitar um palpite e oferecer a possibilidade de sair do jogo:

`Digite sua jogada (letra ou palavra), ou \q para sair:`. A sequência de caracteres especial para sair do jogo é \q.

Ao receber uma mensagem `GAMEOVER`, a aplicação cliente deve exibir `O jogo terminou.` Em seguida, no caso de vitória, deve exibir `A palavra <palavra> foi adivinhada por <nome>!`, ou, no caso de derrota, deve exibir `A palavra <palavra> não foi adivinhada. Último palpite por: <jogador>`. Por fim, o cliente deve encerrar a conexão com o servidor e terminar a execução.

Se ocorrer algum erro de execução, conforme explicado na Seção 3.1, basta imprimir que o erro ocorreu na saída padrão. Dependendo do erro, a execução deve prosseguir ou ser terminada.

Veja um exemplo de execução da aplicação cliente quando é jogador comum:

```
$ ./hangman-client Glenn 192.168.137.202:9999

Conectando ao servidor...
Aguardando o jogo comecar...
Jogo iniciado!
Vidas para adivinhar: 7
Tamanho da palavra: 4 letras.


 -----
 |   |
 |
 |
 |
 |
------  _ _ _ _

Digite sua jogada (letra ou palavra), ou \q para sair: a
jogador Glenn fez uma jogada: a. Restam 7 vidas.


 -----
 |   |
 |
 |
 |
 |
------  _ a _ a
Letras erradas:

Digite sua jogada (letra ou palavra), ou \q para sair: b
jogador Glenn fez uma jogada: b. Restam 6 vidas.


 -----
 |   |
 |   o
 |
 |
 |
------  _ a _ a
Letras erradas: b
```

```
Digite sua jogada (letra ou palavra), ou \q para sair: l
jogador Glenn fez uma jogada: l. Restam 6 vidas.

 -----
 |   |
 |   o
 |
 |
 |
------  l a _ a
Letras erradas: b

Digite sua jogada (letra ou palavra), ou \q para sair: p
jogador Glenn fez uma jogada: p. Restam 5 vidas.

 -----
 |   |
 |   o
 |   |
 |
 |
------  l a _ a
Letras erradas: b p

Digite sua jogada (letra ou palavra), ou \q para sair: v
O jogo terminou.
A palavra 'lava' foi adivinhada por Glenn!
Encerrando conexao com o servidor...
```

E um exemplo quando o cliente é jogador mestre:

```
$ ./hangman-client Magus localhost:9999

Conectando ao servidor...
Aguardando o jogo comecar...
Voce e' o mestre do jogo!
Digite a palavra: Lava

Jogo iniciado!
Vidas para adivinhar: 7
Tamanho da palavra: 4 letras.

 -----
 |   |
 |
 |
 |
 |
------  _ _ _ _

jogador Glenn fez uma jogada: a. Restam 7 vidas.
```

```
 -----
 |   |
 |
 |
 |
 |
------  _ a _ a
Letras erradas:

jogador Glenn fez uma jogada: b. Restam 6 vidas.

 -----
 |   |
 |   o
 |
 |
 |
------  _ a _ a
Letras erradas: b

jogador Glenn fez uma jogada: l. Restam 6 vidas.

 -----
 |   |
 |   o
 |
 |
 |
------  l a _ a
Letras erradas: b

jogador Glenn fez uma jogada: p. Restam 5 vidas.

 -----
 |   |
 |   o
 |   |
 |
 |
------  l a _ a
Letras erradas: b p

O jogo terminou.
A palavra 'lava' foi adivinhada por Glenn!
Encerrando conexao com o servidor...
```

6 Dicas

Esta seção contém algumas dicas para o desenvolvimento do EP.

6.1 Um send não necessariamente equivale a um receive

É preciso entender a abstração de envio e recebimento de mensagens provida pela interface do socket. Na prática, o TCP opera em fluxos (streams) de dados, não em pacotes bem definidos.

Não há como enviar uma mensagem de aplicação com começo e fim bem definidos via TCP. Em vez disso, existem dois fluxos em uma conexão TCP: um fluxo de entrada e um fluxo de saída. O fluxo de entrada é obtido através de uma função de *receber* (receive), e pode-se escrever no fluxo de saída através de uma função de *enviar* (send). Se um lado chama *enviar* para enviar 5 bytes, e depois chama *enviar* para enviar mais 5 bytes, então há 10 bytes que são colocados no fluxo de saída. O lado receptor pode decidir ler os bytes um por um de seu fluxo de recebimento (chamando *receber* 10 vezes), ou pode tentar ler todos os 10 bytes de uma vez com uma única chamada de *receber*.

Enviar dados para o fluxo TCP é bastante fácil, tudo o que se precisa fazer é chamar *enviar*, e os bytes apropriados são enfileirados no fluxo de saída. Receber dados do fluxo TCP é um pouco mais complicado, porque a operação *receber N bytes* esperará até que pelo menos um byte e no máximo N bytes cheguem no fluxo de entrada antes de retornar. Note que a operação *receber N bytes* será concluída mesmo que nem todos os N bytes sejam lidos, dando à aplicação a chance de agir sobre dados parciais enquanto o restante dos bytes de dados está em trânsito. No mundo real, muito poucos programas conseguem processar recebimentos parciais; quase todos os programas precisam de um buffer para armazenar recebimentos parciais até que tenham dados suficientes para fazer um trabalho significativo. O contrário também é possível: nada impede que o resultado de *receber N bytes* contenha mais de uma mensagem de aplicação, se as mensagens forem pequenas e N for grande.

Resumindo: TCP opera em fluxos, não em pacotes. No entanto, a maioria dos protocolos de aplicação, incluindo o especificado neste exercício, são baseados na ideia de mensagens. Como o TCP opera em fluxos, é preciso escrever código adicional para encapsular as mensagens enviadas e recebidas de forma adequada.

6.2 Ferramentas

Algumas ferramentas são úteis para depurar aplicações de rede. Uma delas é o Wireshark, que permite observar os pacotes que são transmitidos nas interfaces de rede, podendo ser usada para monitorar a comunicação entre o cliente e o servidor.

Outra ferramenta é o telnet, que basicamente funciona como um cliente ou servidor TCP, permitindo enviar dados arbitrários (digitados pelo usuário) na conexão. Usando o telnet, você pode testar a interação entre cliente e servidor de forma separada, o que pode facilitar a depuração.

6.3 Testes que envolvem aleatoriedade e entrada manual de dados

Realizar testes quando a execução do programa depende de um valor aleatório pode ser bastante tedioso. Portanto, pode ser uma boa ideia fixar a escolha do jogador mestre para testes.

Outra tarefa tediosa é digitar dados de teste manualmente. Uma forma simples de evitar a digitação é fazer com que a entrada padrão seja feita através de um arquivo em vez do teclado com o operador <.

Por exemplo, os exemplos da Seção 5 poderiam ser executados da seguinte forma:

```
$ ./hangman-client Glenn 192.168.137.202:9999 < gleen_in.txt
$ ./hangman-client Magus localhost:9999 < magus_in.txt
```

Para refletir os exemplos, o conteúdo dos arquivos seria:

`magus_in.txt`:

```
lava

```

`glenn_in.txt`:

```
a
b
l
p
v

```

7 Implementação

O exercício pode ser realizado individualmente ou em dupla. **Independe da escolha, os grupos devem ser informados no eDisciplinas na atividade** "Escolha de dupla EP".

Sugere-se que seja usada a linguagem de programação python e criados dois scripts: hangman-client.py e hangman-server.py. Porém, os integrantes do grupo podem escolher a linguagem de programação que desejarem para implementar o exercício programa, desde que a especificação fornecida seja seguida. Como consequência, será necessário incluir instruções detalhadas de como compilar e executar o código fonte entregue.

Considere que os programas serão testados em um ambiente Linux Ubuntu 22.04. Portanto, se você for usar algum recurso específico, **considere as versões de compiladores, bibliotecas e interpretadores** disponíveis nesta distribuição. Parte da correção depende da execução da aplicação. Portanto, a nota final do exercício será afetada significativamente se as instruções para a compilação e execução do programa não puderem **facilmente** ser seguidas.

Decisões a respeito do paradigma de programação, organização de código e decisões de projeto em geral ficam a cargo dos integrantes do grupo, e devem ser justificadas no relatório.

8 Entrega e critérios de avaliação

A entrega deve ser feita no eDisciplinas na data indicada. É suficiente que apenas um integrante da dupla faça a submissão. A entrega deve conter, dentro de um arquivo zip, um relatório em formato PDF, o código fonte e explicações detalhadas de como compilar e executar o código.

O relatório deve conter as decisões de projeto feitas pelos integrantes do grupo. Exemplos de algumas perguntas interessantes a serem respondidas no relatório:

- Quais foram as maiores dificuldades enfrentadas?
- Foi usado algum método para o desenvolvimento?
- Quais as estruturas de dados usadas e por quê?
- Como o código foi testado?
- Inclua outras informações que achar relevantes.

Algumas reflexões adicionais a respeito do exercício:

- O que seria necessário modificar para permitir palavras com hífen?
- O que acontece se mais jogadores do que esperado tentarem se conectar ao servidor?
- Como você melhoraria o protocolo?

A nota do EP será atribuída de acordo com três critérios: testes de funcionalidade, aderência à especificação do protocolo e relatório (clareza, objetividade e completude).

8.1 Sobre uso de IA

O objetivo deste exercício não é só produzir um código funcional, mas também se familiarizar com a programação de aplicações em rede.

As ferramentas de IA podem ser poderosos recursos de aprendizado, e seu uso é permitido, contanto que seja feito de forma responsável e ética.

Você pode usar estas ferramentas de IA: 1) se estiver "travado"em um problema específico; 2) precisar de ajuda para entender alguma mensagem de erro ou identificar um possível bug em um trecho específico de código; 3) para refatorar um trecho de código que você já escreveu, visando melhorar a legibilidade ou a eficiência; 4) para complementar casos de teste.

Não use a IA para gerar a solução completa do trabalho. Copiar e colar a solução gerada pela IA, sem entendê-la ou modificá-la, é considerado plágio. A ideia é que você crie o código, e não a IA.

Você deve declarar o uso de qualquer ferramenta de IA no relatório. Informe o nome da ferramenta de IA utilizada e uma breve descrição de como a ferramenta foi usada e os prompts exatos que você usou.

Lembre-se: o objetivo é que você aprenda a pensar como um programador de redes. A IA é uma ferramenta para te auxiliar nessa jornada, não para fazer o trabalho por você.

Bom exercício!