

# Trabalho prático 2 - Aviãozinho

Entrega individual

Data de Entrega: 22 de junho de 2025, às 23:59

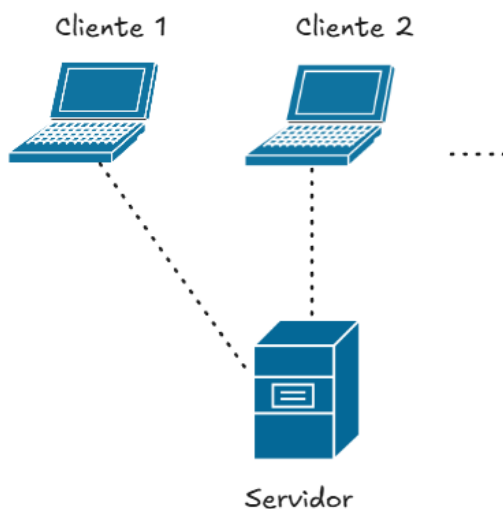
## 1 – Introdução

O **Aviator** (popularmente chamado de “jogo do aviãozinho”) é um jogo de cassino on-line em que um avião decola e o multiplicador de apostas ( $m$ ) sobe continuamente ( $1.00 \times$ ,  $1.01 \times$ ,  $1.02 \times \dots$ ) até “explodir”. Cada jogador aposta um valor antes da decolagem e pode solicitar **cash-out** a qualquer momento durante o voo: o pagamento é  $\text{aposta} \times m$  do instante do pedido. Se o avião explode antes do cash-out, o jogador perde a aposta.



## 2 – Objetivos

- Utilizar apenas as funcionalidades da biblioteca de **sockets POSIX** na **linguagem C** e a comunicação via **protocolo TCP**.
- Implementar um **servidor multithread** capaz de aceitar e gerenciar múltiplas conexões simultâneas (**até 10 jogadores**).



- Implementar um **cliente** que se conecte ao servidor, envie apostas, efetue cash-out e receba atualizações em tempo real.
- Exibir logs padronizados no servidor e nos clientes.
- Garantir compatibilidade com **IPv4** e **IPv6**.

### 3 – Protocolo de comunicação

Todas as mensagens são trocadas pelo socket TCP entre servidor e clientes utilizando a seguinte estrutura:

```
#define STR_LEN 11

struct aviator_msg {
    int32_t    player_id;
    float      value;
    char       type[STR_LEN];
    float      player_profit;
    float      house_profit;
};
```

#### Significado dos campos

- **player\_id**: Um valor  $\geq 1$  atribuído pelo servidor, único por conexão, utilizado em todas as mensagens seguintes para identificar o jogador.
- **value**: Campo numérico em ponto flutuante (**float**), exibido sempre com **duas casas decimais**. O significado específico depende do type e está detalhado na tabela abaixo. Quando especificado como “-”, o campo não é utilizado e pode ser preenchido com zero ou qualquer valor padrão.
- **player\_profit**: Profit acumulado do jogador. Indica o saldo de ganhos ou perdas do jogador durante a sessão.
- **house\_profit**: Profit acumulado do servidor (casa). Representa o saldo total de ganhos ou perdas do servidor em relação a todos os jogadores conectados na sessão.

type	Origem	value	Descrição
start	Servidor → Cliente	Cronômetro de 10 segundos	Marca o começo da rodada e realiza contagem regressiva.
closed	Servidor → Cliente	-	Janela de apostas encerrada.
bet	Cliente → Servidor	Valor da aposta	Enviado antes da rodada começar: só uma aposta por rodada.
cashout	Cliente → Servidor	-	Pedido de saque. O servidor calcula o multiplicador atual e responde.
multiplier	Servidor → Cliente	Multiplicador atual $m(t)$	Broadcast a cada 100 ms durante o voo.
explode	Servidor → Cliente	Multiplicador final $m_e$	Indica explosão do avião; apostas ainda não sacadas são perdidas.
payout	Servidor → Cliente	Valor pago	Pagamento ao cliente após um saque bem sucedido.
profit	Servidor → Cliente	Profit acumulado	Enviado após cashout/explode para informar profit atualizado.
bye	Bidirecional (C→S ou S→C)	-	Encerramento da conexão.

## 4 – Inicialização dos programas

### Servidor

```
./server <v4|v6> <port>
```

O servidor exige dois argumentos:

- **<protocol>**: v4 (IPv4) | v6 (IPv6)
- **<port>**: A porta na qual o servidor está ouvindo.

Exemplo: `./server v4 51511`

### Cliente

```
./client <server_ip> <port> -nick <apelido>
```

O cliente exige três argumentos:

- **<server\_ip>**: O IP (ou hostname) do servidor (por exemplo, 127.0.0.1 se estiver na mesma máquina).
- **<port>**: A porta na qual o servidor está ouvindo (deve ser igual à usada no servidor).
- **<apelido>**: Identificação do jogador.

Exemplo: `./client 127.0.0.1 51511 -nick Flip`

**Lista de erros obrigatórios a serem tratados no cliente, em ordem de prioridade:**

Descrição do erro	Exemplo de comando	Mensagem
Número de argumentos incorreto	<code>./client 127.0.0.1 51511 -nick Flip -bet 10</code>	Error: Invalid number of arguments
Flag <code>-nick</code> ausente	<code>./client 127.0.0.1 51511 Flip</code>	Error: Expected '-nick' argument
Apelido muito longo (> 13)	<code>./client 127.0.0.1 51511 -nick FlipFlopFlaFlu</code>	Error: Nickname too long (max 13)
Valor de aposta inválido	<code>-10</code> ou <code>0</code>	Error: Invalid bet value
Caractere inválido	Diferente de "C" / Diferente de "Q"	Error: Invalid command

OBS: Não é necessário tratar erros fora da lista.

## 5 – Cálculo do ponto de explosão

A explosão do avião ocorre em um ponto calculado **dinamicamente a cada rodada**, com base no número de jogadores ativos e no total apostado. O objetivo é garantir uma **variação natural de multiplicadores** entre rodadas, ao mesmo tempo em que se preserva a vantagem da casa.

O ponto de explosão é definido pelo **multiplicador máximo da rodada**, denotado por  $m_e$ . Este valor é calculado pelo servidor segundo a seguinte fórmula:

$$m_e = (1 + N + 0.01 \cdot V)^\gamma$$

Onde:

- $N$  é o número de jogadores que apostaram na rodada;

- $V$  é o valor total apostado na rodada, calculado como o somatório das apostas individuais de todos os jogadores:

$$V = \sum_{i=1}^N A_i$$

com  $A_i$  sendo o valor apostado pelo jogador  $i$ ;

- $\gamma$  é o expoente de suavização, que define o crescimento do multiplicador de explosão. **Por padrão, neste trabalho, será utilizado:**  $\gamma = 0.5$

ou seja, aplicaremos a **raiz quadrada** da expressão interna;

- A constante 0.01 é utilizada para **normalizar** o valor monetário das apostas, de modo que seu peso na fórmula seja proporcional ao número de jogadores.

Substituindo  $\gamma = 0.5$ , a fórmula final da explosão torna-se:

$$m_e = \sqrt{1 + N + 0.01 \cdot \sum_{i=1}^N A_i}$$

Este será o **limite máximo do multiplicador**. O avião “explode” assim que o valor atual do multiplicador enviado pelo servidor atinge ou ultrapassa  $m_e$ . Após a explosão, todas as apostas que não realizaram cashout são consideradas perdidas.

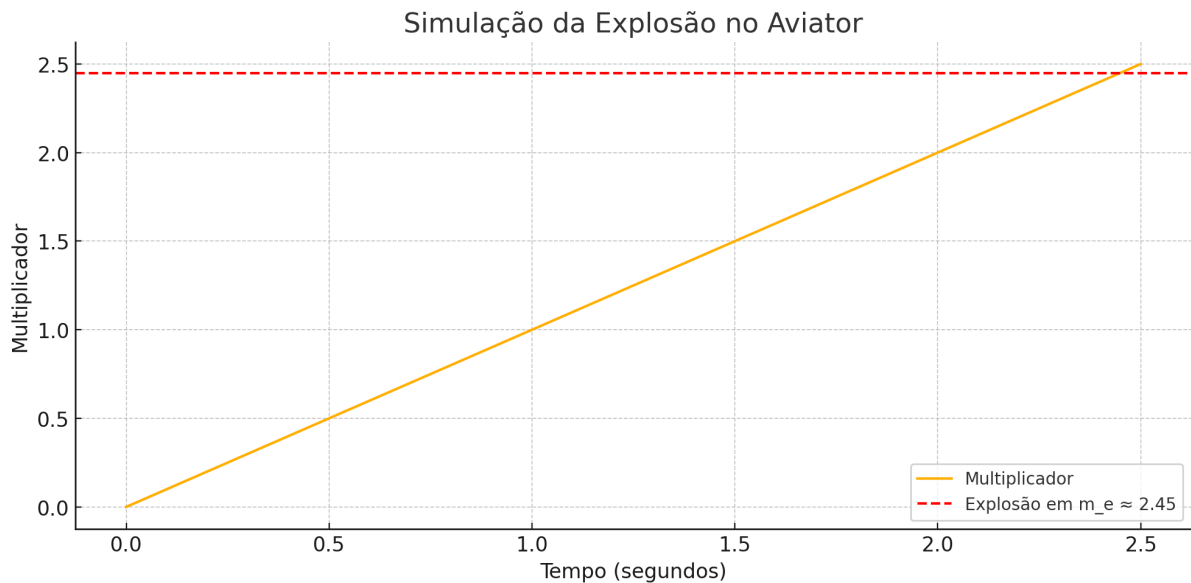
## Exemplo

Se houver 4 jogadores e as apostas forem R\$10, R\$25, R\$15 e R\$50:

- $N = 4, \quad V = 10 + 25 + 15 + 50 = 100$

$$m_e = \sqrt{1 + 4 + 0.01 \cdot 100} = \sqrt{1 + 4 + 1} = \sqrt{6} \approx 2.45$$

O avião irá explodir quando o multiplicador atingir **aproximadamente 2.45x**.



O gráfico acima ilustra o crescimento linear do multiplicador ao longo do tempo. A linha vermelha tracejada marca o exato momento em que o avião explode, encerrando a rodada. Neste exemplo, a explosão ocorre quando o multiplicador atinge aproximadamente  $m_e \approx 2,45$ , valor previamente calculado com base no número de jogadores e no total apostado.

Esse ponto de explosão é obtido pela seguinte fórmula:

$$m_e = \sqrt{1 + N + 0.01 \cdot V}$$

Todos os jogadores que não realizarem o *cashout* até esse instante perdem suas apostas. Por outro lado, aqueles que sacarem antes recebem como pagamento o valor apostado multiplicado pelo valor do multiplicador vigente no momento da solicitação.

## 6 – Cálculo de lucro (profit)

**Cliente:**

- Jogador **saca** antes do aviãozinho explodir:  $profit+ = (payout - bet)$
- Aviãozinho **explodiu** antes do jogador sacar:  $profit- = bet$

## Servidor:

- $ProfitdoServidor = TotalRecebido - TotalPago$

## Exemplo:

**Rodada 1:** aposta 100,00, ganha 200,00

- profit **jogador** = 100,00
- profit **casa** = -100,00 (recebeu 100,00, pagou 200,00 → -100,00)

**Rodada 2:** aposta 100,00, perde

- profit **jogador** = 0,00 (100,00 - 100,00)
- profit **casa** = 0,00 (-100,00 + 100,00)

**Rodada 3:** aposta 100,00, perde

- profit **jogador** = -100,00 (0,00 - 100,00)
- profit **casa** = 100,00 (0,00 + 100,00)

**Rodada 4:** aposta 300,00, ganha 400,00

- profit **jogador** = 0,00 (-100,00 + (400,00 - 300,00))
- profit **casa** = 0,00 (100,00 - (400,00 - 300,00))

## 7 – Fluxo de uma Rodada

1. **Coleta de apostas:** O servidor aguardará até que ao menos um cliente se conecte. Assim que isso ocorrer, uma contagem regressiva de 10 segundos será iniciada.
2. **Cálculo do ponto de explosão:** ao fechar a janela de apostas, calcula-se  $m_e$  com a fórmula acima.



3. **Decolagem (Broadcast):** um loop interno envia mensagens `multiplier` a cada **100 ms** com incremento  $m(t + \Delta t) = m(t) + 0.01$  iniciando em  $m(0) = 1.00$ .
4. **Cash-out:** quando o servidor recebe mensagem cashout de um cliente antes da explosão, responde com o pagamento **apostado** × **multiplicador** e exibe os **profits atualizados**.
5. **Explosão:** quando  $m \geq m_e$ , o aviãozinho explode para todos.
  - a. Para os jogadores que **não realizaram cashout**, suas apostas são perdidas e é exibido o profit individual atualizado para cada um, além do profit da casa.
  - b. Para os jogadores que já haviam sacado, apenas o profit da casa é exibido após a explosão.
6. Repete o fluxo (1-5)
7. **(Alternativo) Saída:** A desconexão pode ser iniciada tanto pelo cliente quanto pelo servidor. (Em ambos os casos, o servidor registra o encerramento nos logs.)
  - a. Quando o cliente envia bye ou fecha o socket, o servidor registra a desconexão e encerra a thread correspondente.
  - b. Quando o servidor encerra, ele envia bye a todos os clientes conectados, que então finalizam suas conexões e exibem a mensagem de término.

**OBS:** O profit será perdido quando o cliente ou servidor desconectar (sem persistência).

## 8 – Exibição de logs no terminal do Servidor

O formato abaixo ilustra todos os campos possíveis que podem ser registrados em uma linha de log do servidor. **Na prática, a cada evento, apenas os campos relevantes são exibidos.** Por exemplo, em eventos de aposta, são destacados o

identificador do jogador (id), valor apostado (bet), número de jogadores (N) e valor total acumulado (V).

```
event=<evento> | id=<id_jogador ou *> | m=<multiplicador> |  
me=<ponto_explosao> | N=<numero_de_jogadores> |  
V=<valor_total_apostado> | bet=<valor_apostado> |  
payout=<valor_recebido> |  
player_profit=<lucro_acumulado_do_jogador> |  
house_profit=<lucro_acumulado_do_servidor>
```

Os eventos registrados são: início de rodada, fechamento da janela de apostas, aposta realizada, cashout, atualização de multiplicador, explosão do aviãozinho, pagamento, lucro e encerramento de conexões. A tabela abaixo lista todos os possíveis tipos de evento registrados nos logs do servidor:

event
start
closed
bet
cashout
multiplier
explode
payout
profit
bye

Exemplo de logs do **servidor**:

**asterisco (\*)** representa todos os clientes.

**Início** da rodada:

```
event=start | id=* | N=3
```

**Recebendo apostas** de jogadores:

```
event=bet | id=1 | bet=20.00 | N=1 | V=20.00  
event=bet | id=2 | bet=30.00 | N=2 | V=50.00  
event=bet | id=3 | bet=20.00 | N=3 | V=70.00
```

**Apostas encerradas:**

```
event=closed | id=* | N=3 | V=70.00
```

**Multiplicador** durante o voo:

```
event=multiplier | id=* | m=1.01  
event=multiplier | id=* | m=1.02  
event=multiplier | id=* | m=1.03  
.  
.  
.  
event=multiplier | id=* | m=1.45
```

**Cashout** bem-sucedido de um jogador (exemplo):

```
event=cashout | id=2 | m=1.45  
event=payout | id=2 | payout=43.50  
event=profit | id=2 | player_profit=13.50
```

Continuação dos multiplicadores após cashout:

```
event=multiplier | id=* | m=1.46  
.  
.  
.  
event=multiplier | id=* | m=2.45
```

**Explosão do avião:**

```
event=explode | id=* | m=2.45
event=explode | id=1 | m=2.45
event=profit | id=1 | player_profit=-20.00
event=explode | id=3 | m=2.45
event=profit | id=3 | player_profit=-20.00
event=profit | id=* | house_profit=26.50
```

Caso o jogador de id=1 emita o comando “Q” para **desconectar** (bye):

```
event=bye | id=1
```

Caso o servidor emita o comando “Q” para se **desconectar** (bye):

```
event=bye | id=*
```

## 9 – Exibição de logs no terminal do Cliente

### 1. Antes de apostar

```
Rodada aberta! Digite o valor da aposta ou digite [Q] para sair (X
segundos restantes):
```

#### 1.1. Logo após apostar

```
Aposta recebida: R$ XX,XX
```

### 2. Após encerrar as apostas

```
Apostas encerradas! Não é mais possível apostar nesta rodada.
```

#### 2.1. Se tiver apostado

```
Digite [C] para sacar.
```

#### 2.2. Atualização periódica de multiplicador

Multiplicador atual: 1.00x  
Multiplicador atual: 1.01x  
Multiplicador atual: 1.02x  
...

### 2.3. Aviãozinho explodiu

Aviãozinho explodiu em: Y.YYx

#### 2.3.1. Sacou antes do aviãozinho explodir

Você sacou em Y.YYx e ganhou R\$ XX,XX!

#### 2.3.2. O aviãozinho explodiu antes de sacar

Você perdeu R\$ XX,XX. Tente novamente na próxima rodada! Aviãozinho tá pagando :)

#### 2.3.3. Profit atualizado (após sacar ou explodir sem sacar)

Profit atual: R\$ XX,XX

#### 2.3.4. Profit atualizado do servidor (após o aviãozinho explodir)

Profit da casa: R\$ XX,XX

### 3. Após parar de jogar (saiu)

Aposte com responsabilidade. A plataforma é nova e tá com horário bugado. Volte logo, <nome>.

#### 3.1. Encerramento do servidor (saiu)

O servidor caiu, mas sua esperança pode continuar de pé. Até breve!

**Sempre que um cliente é desconectado, seja por solicitação do próprio cliente ou pelo encerramento do servidor, o servidor deve:**

- Encerrar a thread dedicada ao cliente;
- Remover o jogador das listas internas (apostas e jogadores ativos);
- Registrar o evento correspondente nos logs.

## Como gerenciar clientes que se conectam em momentos distintos:

- **Caso 1:** Cliente conectado antes do servidor anunciar uma nova rodada:

Ao anunciar nova rodada, servidor envia start com a contagem regressiva (**10s**):

```
Rodada aberta! Digite o valor da aposta ou digite [Q] para sair (10 segundos restantes):
```

- **Caso 2:** Cliente conectado durante a janela de apostas aberta:

Imediatamente após a conexão, o servidor informa o estado atual com o tempo restante, por exemplo 6 segundos restantes:

```
Rodada aberta! Digite o valor da aposta ou digite [Q] para sair (6 segundos restantes):
```

- **Caso 3:** Cliente conectado após apostas fechadas (durante o voo do aviãozinho):

O servidor informa imediatamente após a conexão que as apostas estão encerradas:

```
Apostas encerradas! Não é mais possível apostar nesta rodada.
```

## 10 – Exemplo de Execução ( $m_e = 2.00$ )

Terminal servidor	Terminal Jogador 1	Terminal Jogador 2
<code>./server v4 51511</code>		
<code>event=start   id=*   N=1</code>	<code>./client 127.0.0.1 51511 -nick Flip</code>	
	Rodada aberta! Digite o valor da aposta ou digite [Q] para sair (10 segundos restantes): 50	
<code>event=bet   id=1   bet=50.00   N=2   V=50.00</code>	Aposta recebida: R\$ 50,00	<code>./client 127.0.0.1 51511 -nick Flop</code>

		Rodada aberta! Digite o valor da aposta ou digite [Q] para sair (7 segundos restantes): 50
event=bet   id=2   bet=50.00   N=2   V=100.00		Aposta recebida: R\$ 50,00
event=closed   id=*   N=2   V=100.00	Apostas encerradas! Não é mais possível apostar nesta rodada. Digite [C] para sacar.	Apostas encerradas! Não é mais possível apostar nesta rodada. Digite [C] para sacar.
event=multiplier   id=*   m=1.01  event=multiplier   id=*   m=1.02 ... .	Multiplicador atual: 1.01x Multiplicador atual: 1.02x ... , Multiplicador atual: 1.69x  ... .	
event=cashout   id=1   m=1.70  event=payout   id=1   payout=85.00  event=profit   id=1   player_profit=35.00	C Você sacou em 1.70x e ganhou R\$ 85,00! Profit atual: R\$ 35,00	
... . event=multiplier   id=*   m=1.99		
event=explode   id=*   m=2.00  event=explode   id=2   m=2.00  event=profit   id=2   player_profit=-50.00  event=profit   id=*   house_profit=15.00	Aviãozinho explodiu em: 2.00x Profit da casa: R\$ 15,00	Aviãozinho explodiu em: 2.00x Você perdeu R\$ 50,00. Tente novamente na próxima rodada! Aviãozinho tá pagando :) Profit atual: R\$ -50,00 Profit da casa: R\$ 15,00
event=start   id=*   N=2		

	Rodada aberta! Digite o valor da aposta ou digite [Q] para sair (Você tem 10 segundos para apostar): X	Rodada aberta! Digite o valor da aposta ou digite [Q] para sair (Você tem 10 segundos para apostar): Q
event=bye   id=2	Error: Invalid command	Aposte com responsabilidade. A plataforma é nova e tá com horário bugado. Volte logo, Flop.
	Rodada aberta! Digite o valor da aposta ou digite [Q] para sair (Você tem 8 segundos para apostar):	
Q event=bye   id=*	O servidor caiu, mas sua esperança pode continuar de pé. Até breve!	O servidor caiu, mas sua esperança pode continuar de pé. Até breve!

## 11 – Desenvolvimento

O projeto requer a implementação de dois componentes essenciais: um servidor e um cliente, ambos baseados no protocolo TCP. É crucial garantir que ambos os componentes sejam compatíveis **tanto com o IPv4 quanto com o IPv6**, proporcionando flexibilidade na escolha do endereço IP.

Para configurar o servidor corretamente, este deve ser capaz de receber, seguindo rigorosamente essa ordem, o tipo de endereço desejado (v4 para IPv4 ou v6 para IPv6) e um número de porta especificado na linha de comando. Recomenda-se a utilização da porta 51511 para manter a padronização do projeto.

Da mesma forma, os clientes precisam receber, também rigorosamente nessa ordem, o endereço IP do servidor e o número de porta para estabelecer a conexão com sucesso. Certifique-se de que ambos os componentes sejam configurados de acordo com essas diretrizes para uma integração eficaz e a comunicação adequada entre servidor e cliente. A seguir, um exemplo de execução dos programas em dois terminais distintos:

### IPv4:

- no **terminal 1**: `./server v4 51511`



- no **terminal 2**: `./client 127.0.0.1 51511 -nick Flip`

- no **terminal 3**: `./client 127.0.0.1 51511 -nick Flop`

## IPv6:

- no **terminal 1**: `./server v6 51511`

- no **terminal 2**: `./client ::1 51511 -nick Flip`

- no **terminal 3**: `./client ::1 51511 -nick Flop`

## Sobre Threads:

O uso de threads em servidores é uma abordagem eficiente para lidar com múltiplas conexões simultâneas. Em um servidor multithread, cada cliente conectado é gerenciado por uma thread separada, permitindo que o servidor processe várias requisições de forma paralela. Essa arquitetura tem várias vantagens, com talvez a principal de evitar que a execução de uma operação mais demorada por um cliente bloqueie o atendimento de outros, garantindo maior responsividade do sistema.

No contexto deste trabalho prático, o servidor será responsável por criar uma nova thread para cada cliente que se conecta. Cada thread atuará de forma independente, lidando com as mensagens daquele cliente. Ao mesmo tempo, o servidor principal continuará rodando em sua thread principal, aceitando novas conexões sem interrupções. Essa abordagem permite manter um fluxo contínuo de comunicação entre os clientes e o servidor, assegurando que todas as mensagens sejam tratadas adequadamente, mesmo em cenários com vários clientes conectados simultaneamente. Para aqueles que não conhecem ou desejam relembrar esses conceitos, recomendo essa [playlist](#), em especial os vídeos 5, 6 e 7.

## Materiais para Consulta e Dicas:

- Capítulo 2 e 3 do livro sobre programação com sockets disponibilizado no Moodle;

- [Playlist de programação com sockets do professor Ítalo Cunha](#);
- O guia de programação em rede do Beej (<http://beej.us/guide/bgnet/>) tem bons exemplos de como organizar um servidor;
- Implemente o trabalho por partes. Por exemplo, implemente o tratamento das múltiplas conexões, depois crie os formatos das mensagens e, por fim, trate os tópicos e as especificidades;
- Procure resolver os desafios da maneira mais simples possível;

## 12 – Avaliação

O trabalho deve ser realizado individualmente e **deve ser implementado com a linguagem de programação C (não sendo permitida a utilização de C++)** utilizando somente a biblioteca padrão (interface POSIX de sockets de redes). Deve ser possível executar seu programa no sistema operacional **Linux** e **não deve utilizar bibliotecas Windows, como o winsock**. Seu programa deve interoperar com qualquer outro programa implementando o mesmo protocolo (você pode testar com as implementações dos seus colegas). Procure escrever seu código de maneira clara, com comentários pontuais e bem indentados. Isto facilita a correção dos monitores e tem impacto positivo na avaliação.

Seu servidor será corrigido de forma semi-automática por uma bateria de testes. Cada teste verifica uma funcionalidade específica do servidor.

Para a correção os seguintes testes serão realizados (com IPv4 e IPv6):

Item	Peso
Conexão de múltiplos clientes	10%
Sincronização e concorrência (threads)	15%
Cálculo correto de $m_e$ & difusão do multiplicador	15%
Tratamento de cash-out, explosão, payout e profit	15%
Formato e exatidão dos logs	10%
Fechamento de conexão	10%
Tratativa de erros e argumentos	5%
Documentação	20%

## 13 – Informações importantes

Cada aluno deve entregar documentação em PDF de até 4 páginas (duas folhas), sem capa, utilizando fonte Arial tamanho 12, e figuras de tamanho adequado ao tamanho da fonte. A documentação deve discutir desafios, dificuldades e imprevistos do projeto, bem como as soluções adotadas para os problemas.

**Atenção:** Será dada grande importância a parte da *discussão dos desafios encontrados*, pois as dificuldades ao longo do projeto sempre existem.

A documentação corresponde a 20% dos pontos do trabalho, mas só será considerada para as funcionalidades implementadas corretamente.

**Será adotada média harmônica entre as notas da documentação e da execução, o que implica que a nota final será 0 se uma das partes não for apresentada.**

Cada aluno deve entregar, além da documentação, o código fonte em C e um Makefile para compilação do programa.

Instruções para submissão e compatibilidade com o sistema de correção semi-automática:

- O **Makefile** deve compilar o cliente em um binário chamado “**client**” e o servidor em um binário chamado “**server**” dentro de uma pasta chamada “**bin**” na raiz do projeto.
- **Seu programa deve ser compilado com a execução do comando **make**, ou seja, sem a necessidade de parâmetros adicionais.**
- A entrega deve ser feita no formato ZIP, com o nome seguindo o seguinte padrão: **TP2\_MATRICULA.zip**

## 14 – Desconto de Nota por Atraso


Os trabalhos poderão ser entregues até a meia-noite do dia especificado para a entrega. **Não serão aceitos trabalhos com atraso.**

## 15 – Detecção de Trapaça

O trabalho prático desta disciplina **NÃO aceitará** trechos de código gerados por inteligência artificial, especificamente LLMs (Large Language Models). Para garantir

a integridade acadêmica, utilizaremos um **analisador**, baseado no estado da arte, capaz de:

- Detectar com alta precisão padrões e “impressões digitais” de plataformas como **ChatGPT, DeepSeek, Gemini, Claude e afins**;
- Identificar tentativas de ofuscação ou modificações em código.

 **ATENÇÃO: NÃO HÁ “atalhos” possíveis:** qualquer sinal de código plagiado ou gerado resultará em nota **ZERO**. Sejam criativos, mas acima de tudo, **originais**.