**Aluna:** Gabriela Tavares Barreto
**Matrícula:** 2018074657

# Documentação TP1 - Servidor TCP para Jogo de Labirinto

Este código implementa um servidor TCP que gerencia um jogo de labirinto. Ele processa conexões de clientes, manipula mensagens estruturadas (struct action), e realiza operações no jogo com base nos comandos recebidos. O servidor suporta IPv4 e IPv6.

## Estrutura do Código do Servidor: server.c

## 1.Funções Principais

### 1.1 *main*

**- Propósito:** Configura e gerencia o ciclo de vida do servidor.

**- Passos Principais:**

1. Valida os argumentos passados na linha de comando.
2. Inicializa o estado do jogador (initPlayer) e carrega o labirinto de um arquivo.
3. Cria um socket (socket) e o associa a um endereço e porta (bind).
4. Coloca o socket em modo de escuta (listen) e aceita conexões de clientes (accept).
5. Chama handleClient para processar cada cliente.

### 1.2 *handleClient*

**- Propósito:** Processa as mensagens do cliente e executa ações relacionadas ao jogo.

**- Passos Principais:**

1. Aguarda mensagens do cliente (read).
2. Decodifica a mensagem (memcpy) em uma estrutura struct action.
3. Processa diferentes tipos de comandos:
4. Envia a resposta ao cliente send.
5. Fecha o socket do cliente após encerrar a conexão.

## 2. Fluxo do Servidor:

1. O servidor é iniciado e escuta na porta definida (padrão: 51511).
2. Um cliente conecta, e o servidor imprime *client connected*.
3. O cliente envia comandos (start, move, etc.), que são processados em handleClient.
4. O servidor atualiza o estado do jogo e responde ao cliente.
5. Quando o cliente desconecta ou envia exit, a conexão é encerrada.
6. O servidor continua ativo, esperando por novos clientes.

## 3. Considerações

**- Suporte a IPv4/IPv6:** Configuração dinâmica baseada no argumento <v4/v6>.

**- Restrições:**

- Apenas um cliente é processado por vez.
- O código não suporta múltiplas conexões simultâneas.

**- Robustez:**

- Tratamento básico para desconexão e encerramento de sockets.

**Estrutura do Cliente TCP**

*Descrição Geral de client.c*

Este código implementa um cliente TCP para um jogo de labirinto. Ele se conecta ao servidor, envia comandos do usuário e processa as respostas do servidor, exibindo informações como movimentos válidos, dicas e o estado do labirinto.

**1.Funções e Comportamento**

**1.1 main**

**Propósito:** Estabelece conexão com o servidor e gerencia a interação do cliente durante o jogo.

**Comportamento Principal:**

1. Inicialização da Conexão:
2. Detecta se o endereço do servidor é IPv4 ou IPv6.
3. Configura o socket e tenta se conectar ao servidor.
4. Ciclo de Jogo:
    a. Enquanto o jogo está rodando (gameRunning)
    b. Recebe comandos do usuário (scanf).
    c. Mapeia o comando para um tipo de requisição (mapCommand).
    d. Valida e envia os comandos para o servidor.
    e. Recebe e processa as respostas do servidor, como:
        i. Movimentos válidos.
        ii. Dicas (hint).
        iii. Atualizações no estado do labirinto.
        iv. Reset, exit.

**Desafios de Implementação**

Ao longo do desenvolvimento do servidor e cliente para exploração de labirintos, diversos desafios técnicos e conceituais foram encontrados. Esta seção descreve os principais obstáculos e as soluções adotadas para superá-los.

**1. Comunicação Cliente-Servidor**

- ★ **Desafios Específicos:**
    - Enviar e receber mensagens corretamente no formato struct action.
    - Garantir que o cliente interpretasse os dados recebidos e traduzisse as informações, como os movimentos possíveis e o estado do mapa.
    - Tratar de erros de conexão e desconexão inesperada.

★ **Soluções Adotadas:**
- Uso extensivo de testes com diferentes inputs e situações, como envio de comandos inválidos e desconexões abruptas.

**2. Manutenção do Estado do Jogo no Servidor**

O servidor precisava gerenciar duas representações do labirinto:

1. **Labirinto completo:** incluindo as células ocultas e a saída.
2. **Labirinto visível:** contendo apenas as áreas descobertas pelo jogador.

★ **Desafios Específicos:**
- Sincronizar o estado do jogador com o estado do labirinto.
- Atualizar a visão do jogador de forma eficiente à medida que ele se movia.
- Lidar com os diferentes tipos de ações, como map, move, e reset.

★ **Soluções Adotadas:**
- Uso de duas matrizes separadas: uma para o estado completo e outra para a visão parcial do jogador.
- Funções para calcular a visibilidade com base na posição do jogador e atualizar a matriz parcial dinamicamente.
- Implementação rigorosa de verificações para movimentos válidos e atualização do estado somente quando o comando era válido.

**3. Implementação da Funcionalidade Hint**

★ **Desafios Específicos:**
- Identificar um caminho válido sem precisar calcular o mais curto, mas garantindo que o jogador chegasse à saída.
- Traduzir o caminho encontrado em uma sequência de movimentos (up, down, lef`, right) que pudessem ser enviados ao cliente.
- Tratar situações onde não há caminho até a saída.

★ **Soluções Adotadas:**
- Uso do algoritmo de Busca em Profundidade (DFS) para encontrar qualquer caminho válido até a saída.
- Formatação dos movimentos como um vetor moves preenchido com comandos válidos e zeros nos espaços restantes, conforme o protocolo.

**4. Tratamento de Erros**

★ **Desafios Específicos**
- Detectar comandos inválidos antes de enviá-los ao servidor.
- Lidar com respostas inesperadas ou inconsistentes do servidor.

- Prevenir travamentos caso o jogador tentasse acessar áreas bloqueadas ou realizar ações fora de ordem.

★ **Soluções Adotadas**
- Validação dos comandos no cliente antes de enviá-los.
- Implementação de mensagens de erro detalhadas, como "error: you cannot go this way" e "error: command not found".
- Manutenção de uma lógica clara de estados no cliente para garantir que as ações fossem realizadas na sequência correta.

**5. Compatibilidade IPv4 e IPv6**

★ **Desafio:** Seguir a especificação de compatibilidade tanto para IPv4 quanto IPv6 foi desafiador devido às diferenças na configuração de sockets.

★ **Soluções Adotadas:**
- Implementação de suporte condicional com base no argumento fornecido ao servidor (v4 ou v6).
- Uso de funções específicas da API de sockets POSIX para lidar com os diferentes tipos de endereços.

**6. Integração e Testes**

A integração entre cliente e servidor foi testada com diferentes cenários e inputs para garantir robustez e aderência ao protocolo.

★ **Desafios Específicos:**
- Simular situações reais, como desconexões abruptas e comandos inválidos.
- Verificar a interoperabilidade com outras implementações (conforme solicitado nas diretrizes do trabalho).

★ **Soluções Adotadas:**
- Criação de scripts de teste automatizados para validar os diferentes cenários de uso.
- Execução de testes manuais com labirintos de tamanhos variados, entradas bloqueadas e situações de vitória/reset.

**Considerações Finais**

Este projeto foi uma oportunidade para aplicar conceitos fundamentais de programação em C, redes de computadores e comunicação cliente-servidor. Os desafios enfrentados, especialmente na gestão do estado do jogo e na implementação da funcionalidade de "dica", proporcionaram um aprendizado significativo em termos de lógica e organização de sistemas distribuídos.