Trabalho prático 1 - Jokenboom

Arthur Loureiro Nolasco
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
Belo Horizonte, Brasil
arthurnolasco@ufmg.br

***Abstract*—Este trabalho apresenta o desenvolvimento de um servidor/jogo de Jokenpo, implementado em C com foco em princípios de redes. O sistema implementa um jogo que se comunica entre cliente e servidor e atende às especificações do professor.**

***Index Terms*—redes, cliente, servidor, C, sockets, TCP/IP**

I. Introdução

Este trabalho prático desenvolve um sistema cliente-servidor para o jogo ”Jokenboom”, uma adaptação do Pedra-Papel-Tesoura com cinco opções de jogada (Nuclear Attack, Interceptação de Mísseis, Ciberataque, Bombardeio com Drones e Emprego de Armas Biológicas). A comunicação ocorre via TCP, com o servidor suportando IPv4/IPv6. O sistema gerencia rodadas consecutivas, trata empates, e permite ao cliente jogar novamente ou encerrar a sessão, exibindo um placar final.

A. Contextualização e Motivação

A motivação do projeto é desenvolver um sistema cliente-servidor funcional para partidas de Jokenpo expandido, onde o cliente (humano) joga contra o servidor (computador com escolhas aleatórias). A dinâmica envolve rodadas consecutivas, tratamento de empates, histórico de vitórias e opção de jogar novamente, com comunicação via TCP/IP (IPv4/IPv6).

II. Metodologia

O desenvolvimento do Jokenboom seguiu uma abordagem estruturada:

1) **Análise de Requisitos:** Interpretação detalhada da especificação do trabalho, focando em objetivos, regras, requisitos funcionais (conexão, duelo, resultado, erros, feedback, ”jogar novamente”, encerramento), requisitos não funcionais (C, POSIX sockets, Linux) e protocolo de comunicação.
2) **Configuração do Ambiente:** Utilização do Linux, compilador GCC para C, Make para automação da compilação, e sockets POSIX para comunicação em rede.
3) **Implementação da Solução:** Desenvolvimento modular:

   - **Protocolo (`protocolo.h`):** Definição da estrutura `GameMessage` e do enum `MessageType`, compartilhados entre cliente e servidor.
   - **Servidor (`server.c`):** Lógica de socket TCP (IPv4/IPv6), tratamento iterativo de um cliente por vez, e gerenciamento completo do ciclo de jogo: envio de `MSG_REQUEST`, recebimento de `MSG_RESPONSE`, jogada aleatória do servidor, uso da função `determinar_vencedor`, envio de `MSG_RESULT`, tratamento de empates, atualização de placar, fluxo de ”jogar novamente”, tratamento de erros de entrada com `MSG_ERROR` e re-solicitação, e envio de `MSG_END`.
   - **Cliente (`client.c`):** Implementação da conexão TCP e interface de terminal para o usuário interagir com o jogo

(enviar jogadas, visualizar resultados e mensagens).
- **Módulos Comuns (`common.c`, `common.h`):** Utilização de funções auxiliares pré-existentes para manipulação de endereços de socket e logs.

4) **Compilação e Depuração:** Uso de `Makefile` para compilação. Depuração iterativa de erros de compilação e lógica (ex: inclusões faltantes, correção no tratamento de erro do ”jogar novamente”).
5) **Testes e Validação:** Validação funcional com testes sistemáticos (servidor e cliente em terminais separados), replicando cenários do PDF (conexão, vitória/derrota, empate, ”jogar novamente”, erros de entrada, IPv4/IPv6). Logs do servidor e saídas do cliente foram monitorados para conformidade.

III. Arquitetura do Sistema

A. Visão Geral

O Jokenboom utiliza uma arquitetura cliente-servidor. O servidor gerencia a lógica do jogo e o estado das partidas; o cliente provê a interface para o jogador. A comunicação é via TCP/IP (sockets POSIX em C). O servidor é iterativo (um cliente por vez). A modularidade é dada pelos arquivos: `server.c`, `client.c`, `common.c/h`, e `protocolo.h`.

B. Componentes Principais

Os principais componentes do sistema são:

- **Servidor (`server.c`):** Entidade central que aguarda conexões, gerencia regras, estado do jogo, processa jogadas, e interage com o cliente.
- **Cliente (`client.c`):** Interface para o jogador humano, enviando ações e exibindo resultados/mensagens.
- **Módulo de Protocolo (`protocolo.h`):** Define a estrutura `GameMessage` e os tipos `MessageType` para a comunicação.
- **Módulo Comum (`common.c`, `common.h`):** Funções utilitárias de rede (parsing de endereço, inicialização de socket) para servidor e cliente.

C. Interface de Comunicação (Protocolo)

A comunicação entre cliente e servidor é definida pelo `protocolo.h`, que declara a estrutura `GameMessage` e os tipos `MessageType`. Este arquivo atua como um contrato, especificando o formato e significado dos dados trocados via socket.

D. Modularidade em C

O projeto aplica princípios de modularidade dividindo funcionalidades em arquivos `.c` distintos. Cada módulo expõe interfaces através de arquivos `.h`, ocultando detalhes internos de implementação.

IV. Implementação e Estrutura do Código

Implementado em C com sockets POSIX para comunicação TCP/IP, o código organiza separadamente as responsabilidades do cliente, servidor e funções comuns.

A. Estrutura de Mensagens (`protocolo.h`)

A comunicação usa mensagens estruturadas (`GameMessage`) com um campo `type` (`MessageType`) para identificar a natureza da comunicação.

```
typedef enum {
    MSG_REQUEST, MSG_RESPONSE, MSG_RESULT,
    MSG_PLAY_AGAIN_REQUEST,
    MSG_PLAY_AGAIN_RESPONSE,
    MSG_ERROR, MSG_END
} MessageType;

typedef struct {
    MessageType type;
    int client_action;
    int server_action;
    int result;
    int client_wins;
    int server_wins;
    char message[MSG_SIZE];
} GameMessage;
```

B. Lógica de Determinação do Vencedor (`server.c`)

A função `determinar_vencedor` no servidor aplica as regras do Jokenboom.

```
int determinar_vencedor(...) {
    if (c_act == s_act) return -1;
    if ((c_act==0 && (...)) ||
        (c_act==1 && (...)) ||
        (c_act==2 && (...)) ||
        (c_act==3 && (...)) ||
        (c_act==4 && ...))) return 1;
    return 0;
}
```

V. Funcionalidades Implementadas

O sistema Jokenboom implementa as seguintes funcionalidades chave:

- **Mecânica de Jogo Expandida:** Suporte às cinco opções de jogada e suas regras de vitória/derrota.
- **Comunicação Cliente-Servidor via TCP.**
- **Suporte a IPv4 e IPv6.**
- **Ciclo de Jogo Completo:** Gerenciamento de rodadas, resultados e feedback.
- **Tratamento de Empates:** Re-solicitação automática de jogada.
- **Sistema de Placar por Sessão.**
- **Opção de "Jogar Novamente".**
- **Tratamento de Erros de Entrada do Usuário** (validação no servidor com re-solicitação).

VI. Interface e Resultados das Execuções

A interação com o sistema ocorre via terminal. Exemplos abaixo.

A. Conexão e Início de Jogo

Cliente conecta, servidor aceita, cliente recebe o primeiro prompt.

B. Interação de uma Rodada (Exemplo de Derrota)

Cliente joga, servidor processa e envia resultado.

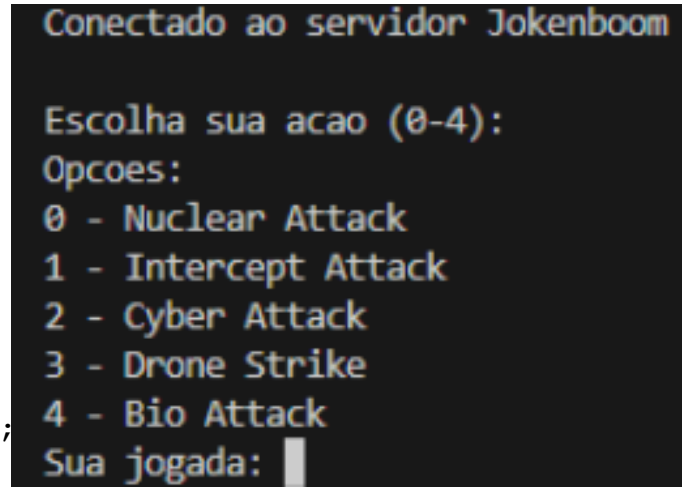

Fig. 1. Cliente conectado e recebendo solicitação de jogada.

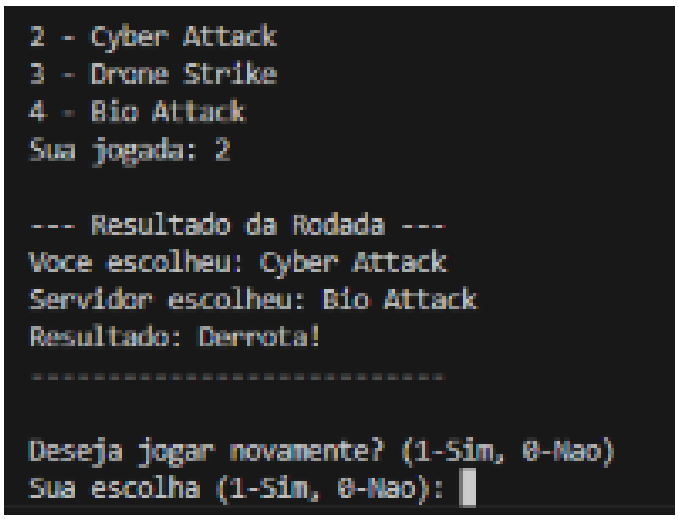

Fig. 2. Exemplo de rodada resultando em derrota para o cliente.

C. Exemplo de Fim de Jogo (Não Jogar Novamente)

Cliente opta por não jogar novamente; servidor envia placar e encerra.

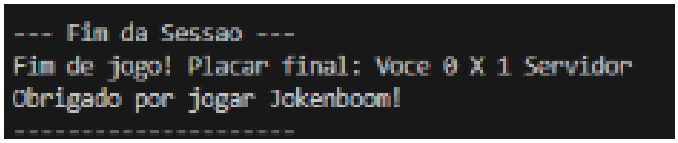

Fig. 3. Fim de sessão após cliente optar por não jogar novamente.

VII. Tratamento de Erros no Projeto

O tratamento de erros focou em falhas de comunicação/rede e entradas inválidas do usuário:

- **Erros de Rede/Socket:** Falhas em operações de socket (criação, bind, etc.) ou desconexões são tratadas com mensagens de erro (`perror`, `logexit`) ou encerramento da sessão do cliente específico, permitindo ao servidor continuar operando para novas conexões.
- **Erros de Entrada do Usuário (Validação no Servidor):**
  - **Jogada Inválida (0-4):** Servidor envia `MSG_ERROR` ("Por favor, selecione um valor de 0 a 4.") e reenvia `MSG_REQUEST`.
  - **Resposta Inválida para "Jogar Novamente" (0/1):** Servidor envia `MSG_ERROR` ("Por favor, digite 1 para jogar novamente ou 0 para encerrar.") e reenvia `MSG_PLAY_AGAIN_REQUEST`.

Este tratamento no servidor guia o cliente a fornecer entradas válidas.

VIII. Dificuldades Encontradas

Durante o desenvolvimento, foram superados alguns desafios:

- **Utilização do Makefile:** Dificuldade inicial na configuração para compilar múltiplos arquivos C e gerenciar dependências, como `common.o`, exigindo pesquisa para automação eficiente.
- **Ambiente de Terminal Linux:** Necessidade de revisão de comandos de terminal e conceitos de PDS2 para gerenciar processos (servidor/cliente) e testes.
- **Gerenciamento de Ponteiros em C:** Desafios na manipulação de ponteiros para estruturas de mensagem (`GameMessage`) e endereços de socket (`sockaddr_storage`, casts, `sizeof`) para comunicação em rede, demandando estudo para evitar erros de segmentação ou corrupção de dados.

A superação destas dificuldades consolidou conhecimentos práticos.

IX. Conclusão

O sistema cliente-servidor Jokenboom foi implementado em C, aplicando conceitos de redes com sockets TCP. O projeto atendeu aos requisitos de funcionalidade, protocolo (IPv4/IPv6) e tratamento de erros. As dificuldades enfrentadas, especialmente com ponteiros e lógica de estado em rede, enriqueceram o aprendizado em programação de sistemas distribuídos.