Relatório - Servidor de Campo Minado

- Ivan Vilaça de assis - ivanassis07@gmail.com - 2021421931

Neste trabalho, foi elaborado um jogo de campo minado simplificado que funciona entre máquinas diferentes através da comunicação de rede. Os desafios foram mais na parte de entender e desenvolver os conceitos de rede na prática do que em elaborar as regras do jogo, visto que estas já vinham bem definidas na especificação.

No decorrer do projeto, lendo a documentação do Beej sobre programação em rede com soquetes, foi possível aprender como funciona o IP e as portas na comunicação em rede e reforçar o conhecimento dos diferentes tipos de protocolo, TCP e UDP, da camada de transporte vistos na disciplina de redes. Além de ter visto vários dos conceitos já vistos em aula, como das camadas de rede e do encapsulamento presente nelas.

A seguir temos os principais desafios encontrados ao longo do desenvolvimento do projeto:

1. **Esquema cliente-servidor**
   O primeiro desafio foi entender melhor como funcionaria a comunicação cliente-servidor no contexto do projeto. Mas os vídeos disponibilizados no Moodle e a aula sobre o trabalho ajudaram com este aspecto, mostrando a ordem em que as operações de abrir e fechar conexão ocorriam e também as de enviar e receber dados entre o cliente e o servidor.
2. **As estruturas e funções para lidar com sockets**
   O conceito de socket, um descritor de arquivos usado para comunicação entre processos através de uma rede ou em uma mesma máquina, foi algo fácil de entender desde o início. Mas entender como utilizá-los dentro do código através das diferentes estruturas disponíveis (*sockaddr_storage, sockaddr_in, sockaddr_in6*) não foi algo simples, principalmente a compreensão do que cada um de seus campos esperava e como era possível fazer a conversão de uma *struct* para outra.

   Buscando resolver essas dúvidas em relação a implementação, a leitura do Guia Beej de programação de redes foi fundamental, visto que nele havia a explicação com exemplos de uso e detalhes mais específicos, como a parte de conversão entre as estruturas.

3. **Controlar o fechamento dos sockets e do servidor**
   Nas especificações foi dito que o servidor deveria receber múltiplas mensagens do cliente sem fechar até que o cliente enviasse o comando “exit” e que ele deveria ficar aberto mesmo após um cliente sair.

Implementar este comportamento foi um pouco trabalhoso porque foi necessário entender bem como funciona a parte de fechar o socket e onde eu deveria usar o comando *close()*. No lado do servidor, eu fiz isso utilizando um *close()* fora de um while que recebia os comandos de um cliente conectado, assim, caso houvesse um “exit”, haveria um *break* que fecharia o socket e a conexão do cliente seriam fechados e o servidor ainda ficaria ativo para receber novos clientes.

No lado do cliente, foi mais fácil visto que era necessário apenas manter a conexão aberta pegando todos seus comandos no decorrer da partida até que ele desse “exit”, quando isso ocorresse o socket dele é fechado e ele também.

Uma parte do trabalho que gerou dúvida foi o assunto de serialização dos dados para enviar nos sockets do cliente e do servidor. Eu não conhecia esta prática, mas me deparei com ela enquanto pesquisava sobre envio de dados em sockets e vi que ela é importante porque garante portabilidade do código, visto que previne problemas que surgem devido a incompatibilidade de representação de dados em diferentes arquiteturas.

Apesar disso, optei por passar a *sctruct action* do jogo usando um ponteiro porque foi dado na especificação e também porque desta forma ficaria possível testar usando os clientes e servidores de outros colegas que seguiram a documentação.

Mesmo não tendo serializado, aprendi que caso eu quisesse fazer o processo , eu teria de serializar cada campo da *struct action* de forma independente antes de enviar. O receptor receberia os dados e teria que saber a priori em qual ordem os campos foram serializados para que ele desserializasse corretamente e pudesse usá-los para atualizar os dados do jogo.

O trabalho foi uma ótima maneira de consolidar os conceitos de rede aprendidos em aula e de ter uma pequena introdução ao mundo de programação em redes.