UNIVERSIDADE FEDERAL DE MINAS GERAIS
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

DCC023 - REDES DE COMPUTADORES

Mônica Emediato Mendes de Oliveira

3 de Setembro de 2020

Trabalho Prático 1

Jogo da forca através de sockets.

Neste trabalho será desenvolvido um jogo da forca por meio de um cliente que envia letras como palpites e um servidor que os recebe e responde os locais de ocorrência da letra, caso exista na palavra.

3 de Setembro de 2020

1 Introdução

O trabalho prático desenvolvido tem o objetivo de trazer conhecimento sobre sockets de comunicação entre dois processos distintos. Esses processos podem estar no mesmo sistema ou em sistemas diferentes, se comunicando então através de um canal de transmissão. Essa comunicação se dá baseada nos padrões e protocolos da arquitetura TCP/IP. Derssa forma, foi desenvolvido um par de programas, um cliente e outro servidor. Em suma, o servidor é iniciado e espera que o cliente se conecte a ele para adivinhar a palavra de um jogo de forca.

2 Desenvolvimento e Metodologia

Para realizar esse trabalho, inicialmente, tem-se em mente que um socket é uma abstração que uma aplicação envia e recebe dados. Diferentes tipos de sockets correspondem a diferentes tipos de protocolos. Esse trabalho irá implementar um jogo de forca em protocolo TCP/IP. O código funciona em IPv4 e IPv6, como especificado.

De acordo com as implementações, deve-se usar 1 byte/8 bits para cada indicador de mensagem. Para isso, no servidor foram usados ”uint8 t”para definir o tipo e tamanho. Para definir quais as posições, usou-se ”uint32 t”, para ter uma segurança de que a memória seria suficiente para esse jogo. Para o cliente, usa-se também ”uint16”para as posições e com adição de um char para captura de caractere do usuário. O char equivale a um byte, o qual é capaz de conter um caracter no local setado para esse caracter.

Em caso de uma palavra muito grande a ser adivinhada no jogo, a única alteração a ser feita seria no tamanho dessas variáveis iniciais. Para isso, poderíamos trocá-las para uint32 ou uint64, as quais definiriam que teriam mais bytes a ser lidos e enviados pelo buffer do cliente e servidor. A mensagem foi previamente definida pelo servidor e foi somente enviada o tamanho dela para o cliente, de forma que o cliente não sabia qual seria a mensagem a ser adivinhada.

O programa entre cliente servidor tem quatro tipos de mensagens, como especificado. Em que cada tipo corresponde a um byte diferente. Todas as mensagens devem conter o primeiro byte indicando o tipo (de acordo com a numeração abaixo), seguidos de sua respectiva mensagem. Foi separado da seguinte forma:

1. Início do jogo: enviada pelo servidor para o cliente após a sua conexão, indicando o número de caracteres da palavra a ser descoberta. Exemplo: o primeiro byte da mensagem deve indicar o tipo (o valor inteiro 1). O segundo byte deve ser

um inteiro sem sinal, indicando o tamanho da palavra a ser descoberta.

2. Palpite: enviada pelo cliente ao servidor, indicando uma letra para ser testada, com o seguinte formato: como na mensagem anterior, o primeiro byte deve indicar o tipo (valor inteiro 2), seguido de 1 byte indicando o caractere em ASCII a ser testado.

3. Resposta: enviada pelo servidor como resposta a um palpite, contém o número (n) de ocorrências do caractere testado (pode ser 0, indicando que não há letra na palavra). Os n bytes seguintes devem indicar cada uma posição de ocorrência de tal caractere.

4. Fim de jogo: enviada pelo servidor quando todas as letras da palavra forem encontradas. Tal mensagem deve enviar somente o tipo (valor inteiro 4).

Para fazer isso, sabe-se que a comunicação só será realizada se cliente e servidor chamarem as funções em ordem correta. A sequência de chamada às funções recv() e send() é definida pelo protocolo de comunicação. O socket TCP precisa estar conectado com outro socket antes de enviar qualquer dado. Nesse sentido, pode-se fazer uma alusão à rede telefônica. Antes de falar, é preciso especificar o número que você deseja e a conexão precisa ocorrer. Se a conexão não for feita, é preciso tentar novamente depois. O cliente inicia a conexão enquanto o servidor espera passivamente por clientes para conexão. Para estabilizar a conexão com o servidor, chama-se a função connect() no socket.

Além disso, o cliente lê através da função: ”read(clientSocket, buffer, sizeof(buffer));”a mensagem da forca para que o usuário possa dar palpites sobre o jogo. A palavra que será adivinhada foi declarada previamente no código do servidor para que os testes fossem mais rápidos.

Houve um tratamento para múltiplas conexões de forma que se permite 5 conexões simultâneas, por meio de uma fila de prioridades na qual a função listen() recebe como parâmetro a quantidade de elementos (conexões) que essa lista terá.

3 Código

Os programas foram testados em um computador Ubuntu 20.04, dessa forma a mesma máquina (localhost 127.0.0.1) serviu como servidor e como cliente. O código usa o protocolo TCP para comunicação e foi implementado para IPv4 e IPv6. Há um arquivo Makefile para compilação dos programas, o qual gera dois executáveis com os nomes ’cliente’ e ’servidor’.

O servidor é executado com o seguinte comando: ./servidor porta

O cliente é executado com o seguinte comando:

./cliente ip-servidor porta

O servidor está especificado para receber IPv6 e IPv4, o cabeçalho para endereçamento permite isso. O cliente foi especificado em IPv4, mas poderia ser usado como IPv6 também.

4 Conclusão

Por meio do TP1, foi possível adquirir muito conhecimento acerca da comunicação entre dois sistemas diferentes, bem como o uso das camadas da arquitetura TCP/IP, umas vez que os sockets trabalham na camada de transporte. Além disso, a execução do trabalho permitiu contato com a linguagem C/C++ e com a ideia de sockets de comunicação. Ao desenvolver uma aplicação para ser utilizada em rede, exigiu-se uma implementação muito bem organizada das funções e procedimentos.

Referências

Donahoo, Michael. Calvert, Kenneth. TCP/IP Sockets in C, Practical Guide for Programmers. Second Edition. 2009.