Trabalho Prático 2 - Redes de Computadores

Luiz Philippe Pereira Amaral

[1]Departamento de Ciência da Computação - Universidade Federal de Minas Gerais (UFMG)

1. Introdução

O trabalho desenvolvido em C representa o conceito de um servidor via rede para comunicação entre múltiplos clientes sobre um protocolo específico. Esse servidor é reponsável pelo gerenciamento das conexões e a troca de mensagens entre as pontas. Durante a conexão, cada cliente representa um equipamento genérico em operação e possui um ID único, baseado nisso, ele pode realizar as seguintes ações:

1. Listar os demais equipamentos conhecidos
2. Solicitar a leitura do valor de um desses equipamentos
3. Desconectar-se da rede

O sistem tem ao todo capacidade para 15 equipamentos e as mensagens de sucesso e erro apropriadas são enviadas para cada situação.

2. Implementação

A implementação utiliza a interface padrão POSIX de sockets do C. Para compilar o programa utilize o comando:

```
$ make
```

Para executar o servidor utilize o comando:

```
$ ./server [PORTA]
```

onde a porta se refere a uma porta disponível do computador. E para conectar um cliente utilize:

```
$ ./equipment [HOST] [PORTA]
```

onde o host e a porta se referem respectivamente ao endereço do servidor onde se deseja conectar e a porta onde o servidor está ouvindo por conexões.

A implementação se distribui em três arquivos principais:

1. server.c: implementa a lógica do servidor
2. equipment.c: implementa a lógica de um equipamento (cliente)
3. common.c: implementa funções e tipos utilitários do sistema

2.1. Servidor

O arquivo server.c define a implementação do servidor. Sua função main inicia o socket utilizado para a abertura de conexões com clientes e aguarda por solicitações em um loop. Quando a solicitação de conexão é recebida, caso o limite de clientes não tenha sido atingido, é realizado um handshake com o cliente e uma thread é criada dedicada a escutar solicitações deste cliente. Essa thread distribui um ID único para esse cliente (para fins de simplicidade, o descritor de arquivo do socket no servidor é utilizado como o ID)

e espera por requisições. Ao receber uma requisição, a thread utiliza funções auxiliares para decodificar a mensagem recebida em 4 campos: tipo da mensagem, id de origem, id de destino e payload. A thread prossegue então para processar a solicitação e realizar as movimentações adequadas.

O handshake do servidor é executa na thread principal e consiste de:

1. Aguardar por uma solicitação do tipo ”solicitação de ingresso”do cliente.
2. Utilizar mecanismos de sincronização para entrar em seção crítica e modificar os valores compartilhados entre as threads.
3. Gerar um ID único para o cliente.
4. Verificar se o limite de clientes não foi atingido.
5. Inserir o cliente no seu registro local.
6. Envia uma resposta com o novo ID ao cliente.
7. Fazer uma transmissão em broadcast do tipo ”solicitação de ingresso”para todos os demais clientes conectados.
8. Iniciar a thread do cliente.
9. Envia uma lista dos demais equipamentos conectados ao novo cliente.

Se qualquer etapa do handshake falha (com exceção da transmissão em broadcast) o cliente é desconectado.

O registro local dos equipamentos é feito *in-memory*, ou seja, não existe persistência de dados entre as seções.

O processamento das solicitações pelo servidor é feito por meio da função *handle_requests*. Ela identifica o tipo da mensagem baseado no seu ID e a despacha para o método específico apropriado. Os dois principais tipos de requisição recebidos pelo servidor durante o loop de comunicação com os clientes são ”solicitação de informação”e ”solicitação de remoção”.

2.1.1. Solicitação de informação

O servidor recebe a solicitação de informação de um equipamento de origem para um equipamento de destino. O servidor verifica se os IDs de equipamentos são válidos, caso negativo, ele responde as mensagens de erro adequadas, caso positivo, ele envia para o cliente de destino uma solicitação de informação e aguarda pela resposta. Após receber a resposta da solicitação de informação do destino, ele verifica novamente se os clientes continuam váldos (novamente enviando respostas de erro em caso negativo) e por fim, responde ao equipamento de origem o valor informado pelo equipamento de destino.

2.1.2. Solicitação de remoção

A solicitação de remoeção é recebida pelo servidor quando um cliente deseja se desconectar da rede. Ao receber essa solicitação, o servidor verifica se o cliente está presente nos seus registros, caso ele não esteja, uma mensagem de erro é respondida, utiliza os mecanismos de sincronização para modificar recursos compartilhados entre as threads e remove o cliente de tais registros. Então uma solicitação de remoção é enviada em broadcast para todos os demais equipamentos da rede. Por fim, sua thread de operação é finalizada.

2.1.3. Registros de operação

O servidor apresenta mensagens de operação em seu terminal nas sequintes situações: ao abrir conexões com novos clientes, ao retornar erro a um cliente por atingir o limite de conexões, ao retornar erro para um cliente por não encontrar o equipamento de origem ou destino informados e ao remover um equipamento.

2.1.4. Estruturas de dados

A principal estrutura de dados do servidor é a que representa os argumentos de uma thread de operação de cliente. Essa estrutura consiste de um indice que localiza o a conexão e a thread do cliente nos registros locais e um descritor de arquivo que indica o socket por onde a comunicação com o cliente é feita.

2.2. Cliente

O cliente tem sua implementação definida no arquivo equipment.c. Ao iniciar sua execução, o cliente abre um socket para comunicação pelo servidor no endereço e porta especificados. Ele então prossegue para realizar o handshake com o servidor onde o cliente recebe seu novo ID atribuido e os uma lista com os IDs dos demais equipamentos da rede. Ao concluir o handshake, o cliente inicia duas threads: uma responsável pelo envio de mensagens ao servidor e outra responsável pela leitura de mensagens recebidas. A thread de envio de mensagens lê a entrada do terminal e envia as mensagens adequadas. A thread de leitura decodifica mensagens com funções auxiliares e executa o processamento necessário.

Do lado do cliente, o handshake é feito por meio dos seguintes passos:

1. Enviar uma solicitação do tipo ”solicitação de ingresso”ao servidor.
2. Aguardar pela resposta contendo seu novo ID atribuído e salvá-lo no registro local.
3. Em caso de erro, imprimir a mensagem adequada e finalizar sua execução.
4. Aguardar por uma mensagem do servidor incluindo a lista de equipamentos conectados e salvá-los no registro local.

2.2.1. Envio de mensagens

A thread de envio de mensagens lê os comandos do teclados e os executa. Os comandos possíveis são: listar equipamentos do registro local, buscar a leitura de um equipamento da rede e desconectar-se.

Ao lsitar os equipamentos, eles são buscados do registro local e seus IDs são exibidos na tela. Ao buscar a leitura de um equipamento, uma solicitação do tipo ”solicitação de informação”é enviada ao servidor contendo seu prórpio ID como origem e o ID do equipamento que se deseja fazer a leitura como destino. Ao desconectar-se, uma solicitação do tipo ”solicitação de remoção”é enviada ao servidor, uma mensagem de sucesso é esparada como resposta, então o socket de comunicação é fechado e ambas as threads são finalizadas.

2.2.2. Recebimento de mensagens

Ao receber uma mensage, a thread de leitura realiza sua decodificação por meio de funções auxiliares e e envia para o método *handle_message*, que por sua vez despacha a mensagem para um método específico de acordo com seu tipo, em geral, informações relacionadas a mensagens recebidas são exibidas na tela.

As possíveis mensagens são remoção de um equipamento do registro local, inclusão de um equipamento no registro local, recebimento da lista de equipamentos da rede, solicitação de leitura, resposta de solicitação de leitura, erro e sucesso.

2.3. Utilitários comuns

Os utilitários comuns são funções, tipos e estruturas genéricos ou compartilhados entre as aplicações de cliente e servidor.

2.3.1. Tipos e estruturas

As definições de tipos utilitários são:

1. equipment_t: tipo do identificador de equipamento (alias para inteiro)
2. message_t: estrutura dos campos de composição de uma mensagem trocada entre cliente e servidor, cada mensagem contém um id, uma origem, um destino, o tamanho em bytes do payload e um payload.
3. server_socket_info: estrutura de informações sobre o socket do servidor incluindo porta, endereço, tamanho em bytes da estrutura de endereço e descritor de arquivo do socket.
4. client_socket_info: estrutura que descreve o socket do cliente e sua conexão com o servidor incluindo os descritores de arquivo dos sockets do cliente e servidor.

As principais funções utilitárias implementadas são:

1. slice: retorna uma ”fatia”de uma string.
2. init_message: aloca a memória e inicializa uma mensagem trocada entre cliente e servidor.
3. set_payload: atribui os campos de payload e tamanho de payload de uma mensagem;
4. destroy_message: libera a memória para uma mensagem.
5. read_input: lê a entrada do terminal.
6. decode_args: decodifica uma sequência de bytes em um objeto do tipo mensagem.
7. encode_args: codifica uma mensagem em uma seqência de bytes.
8. create_server_socket: inicializa um socket de servidor.
9. connect_client: recebe conexão de um cliente em socket de servidor.
10. create_equipment_socket: inicializa um socket de cliente.

3. Conclusão

Embora simples, a implementação do projeto demonstra que é possível construir em tempo hábil um sistema de comunicação de múltiplos clientes em uma linguagem básica utilizando apenas as bibliotecas padrão (a maioria em padrão POSIX) para controle de sockets e threads. Seria possível escalar a operação so servidor com otimizações de hardware e software para suportar maiores quantidades de clientes em grandes aplicações do mundo real, dadas as devidas proporções de tempo de desenvolvimento e esforço.