Trabalho Prático 01 - Redes de Computadores TW
Professor: Marcos Augusto Menezes Vieira

Otávio de Meira Lima
2019054900

01 - Introdução

O gerente de uma indústria siderúrgica necessita de uma solução para aumentar o rendimento de toda a sua linha de produção. Sua ideia foi implantar um sistema para manipular dados de equipamentos durante o processo de produção e consultá-los futuramente por uma Central de Monitoramento. O problema proposto pelo trabalho foi implementar um sistema composto por um servidor, que representa a Estação Remota, e um cliente, que representa a Central de Monitoramento. O cliente deve enviar comandos em forma de mensagens para o servidor com o objetivo de manipular e consultar equipamentos disponíveis pela linha de produção e o servidor deve armazenar os dados dos equipamentos e realizar as manipulações solicitadas pelo cliente.

02 - Implementação

02.1 - Linguagem

Foi utilizada a linguagem C para implementar tanto o cliente quanto o servidor. Ambos utilizaram bibliotecas que disponibilizam funções para estabelecer conexões por meio de endereços IP. O desenvolvimento nessa linguagem é bastante desafiador, já que estruturas úteis para manipulações de dados, como listas e mapas, não estão disponíveis.

02.2 - Detalhes da Execução

Para compilar tanto o servidor quanto o cliente, basta apenas rodar o seguinte comando no terminal:

- **make**

Para executar, é necessário rodar os seguintes comandos em sequência. É necessário executar primeiro o servidor, para disponibilizar a porta, para em seguida executar o cliente.

- IPv4:
    - ./server v4 51511
    - ./client 127.0.0.1 51511
- IPv6:
    - ./server v6 51511
    - ./client ::1 51511

Na execução do servidor, o primeiro parâmetro representa qual será o protocolo utilizado. v4 sinaliza a utilização do protocolo IPv4 e v6 sinaliza a utilização do protocolo IPv6. O segundo parâmetro representa a porta de execução do programa. Ambos, servidor e cliente, devem utilizar a mesma porta para prosseguir com a conexão.

Na execução do cliente, o primeiro parâmetro representa o endereço a ser utilizado.

- 127.0.0.1 para conexão com IPv4
- ::1 para conexão com IPv6

O segundo parâmetro deve ser a porta utilizada.
A fim de padronização do projeto, a porta 51511 foi utilizada em todos os testes.

02.3 - Implementação do Cliente

A Central de Monitoramento pode solicitar ao servidor:

- adição de um novo sensor a um equipamento
- remoção de um sensor já instalado a um equipamento
- listagem de todos os sensores presentes e um equipamento
- leitura de sensores presentes em um equipamento

Primeiramente, foi necessário a implementação de uma função que se conecta com o servidor. Conexão com IPv4 e IPv6 não utilizam as mesmas estruturas de dados. Por exemplo, o endereço do servidor em um protocolo IPv4 é dado pela estrutura **sockaddr_in**, enquanto em IPv6 é dado pela estrutura **sockaddr_in6.** Apesar disso, as implementações são semelhantes, utilizando métodos como socket(), inet_pton() e connect(). O retorno do método implementado **connectToServer()** é o socket que será utilizado na conexão. Ele é um inteiro essencial para o envio e recebimento de mensagens.

```c
if (!strcmp(protocol, "127.0.0.1")){

    struct sockaddr_in serv_addr;
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("\n Socket creation error \n");
        exit(1);
    }

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

    if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr)
        <= 0) {
        printf(
            "\nInvalid address/ Address not supported \n");
        exit(1);
    }

    if (connect(sock, (struct sockaddr*)&serv_addr,
                sizeof(serv_addr))
        < 0) {
        printf("\nConnection Failed \n");
        exit(1);
    }

    return sock;

}
```

```c
else if (!strcmp(protocol, "::1")){

    struct sockaddr_in6 serv_addr;
    if ((sock = socket(AF_INET6, SOCK_STREAM, 0)) < 0) {
        printf("\n Socket creation error \n");
        exit(1);
    }

    serv_addr.sin6_family = AF_INET6;
    serv_addr.sin6_port = htons(PORT);
    serv_addr.sin6_addr = in6addr_any;

    if (inet_pton(AF_INET6, "::1", &serv_addr.sin6_addr)
        <= 0) {
        printf(
            "\nInvalid address/ Address not supported \n");
        exit(1);
    }

    if (connect(sock, (struct sockaddr*)&serv_addr,
                sizeof(serv_addr))
        < 0) {
        printf("\nConnection Failed \n");
        exit(1);
    }

    return sock;
}
```

Imagem 1: implementação da conexão do cliente com o servidor

Após a implementação do método, basta apenas de um loop while, com um método getline() para capturar a mensagem a ser enviada para o servidor pelo terminal. Também é necessário o método recv(), que recebe a mensagem do servidor, para verificar se a conexão ainda existe. Caso contrário, o loop é encerrado, a porta é fechada e o cliente é finalizado. O método send() envia a mensagem capturada no terminal para o servidor.

```
94      //loop utilizado para possibilitar o envio de multiplas mensagens
95      while(1){
96
97          message = (char *)malloc(buff_size * sizeof(char));
98          int count = recv(sock, buffer, buff_size, 0);
99
00          if (count == 0) {
01              // Conexão interrompida.
02              break;
03          }
04
05          characters = getline(&message, &buff_size, stdin);
06          send(sock, message, characters, 0);
07
08          free(message);
09
10      }
11      close (sock);
12      free(message);
```

Imagem 2: implementação do envio das mensagens

02.4 - Implementação do Servidor

O servidor recebe as mensagens enviadas pelo cliente e realiza os comandos que foram solicitados. O método connectToClient() realiza as conexões com o cliente. Semelhante ao que foi descrito na secção anterior, a conexão com IPv4 e IPv6 também possuem implementações diferentes, utilizando as mesmas estruturas descritas anteriormente.

Após a conexão ser estabelecida, a função performAction() recebe um comando, aquele enviado pelo cliente, e realiza o que foi solicitado. Cada ação possui sua implementação, mas em geral, é necessário destrinchar a mensagem em ação, sensores e equipamento. Caso a ação não seja reconhecida, o servidor fecha sua conexão e desconecta o cliente. Ambos os sensores e equipamentos estão representados por seus ID 's, que variam entre 01 e 04 necessariamente. Caso algum ID não for reconhecido, uma mensagem de erro é retornada. Os detalhes de cada ação estão descritos a seguir:

- Adição de sensores
    - Os sensores são adicionados na tabela no formato booleano. Ou seja, se um sensor de ID 03 for adicionado ao equipamento de ID 01, a posição [0][2] na matriz será um booleano com valor verdadeiro. Como a matriz possui indexação com início em 0, é necessário armazenar os ID’s como ID - 1.
    - Na tentativa de adição de um sensor que já foi adicionado, o servidor retorna um erro.
- Remoção de sensores
    - Apenas sensores que possuem valor verdadeiro na tabela podem ser removidos. Sensores com valor falso representam um sensor que não está instalado em certo equipamento.
    - Na tentativa de remoção de um sensor que já não existe, o servidor retorna um erro.
- Listagem de sensores
    - Sensores são listados em ordem crescente de ID. Em caso de solicitação de listagem de sensores de um equipamento que não possui nenhum sensor instalado, o servidor retorna a mensagem “none”.
- Leitura de sensores
    - Ao solicitar a leitura de sensores, o servidor deve retornar um valor aleatório para cada sensor e uma mensagem de erro para sensores inexistentes.

Além das ações listadas acima, também foi solicitada uma ação extra que deve terminar a execução do servidor, fechando as portas disponíveis. Consequentemente, o cliente também será finalizado, já que sua conexão com o servidor já não está mais estabelecida.

```
146  //Executa os comandos especificados
147  void performAction(char *command){
148
149      char * action = strtok(command, " ");
150      removeNewLine(action);
151
152  >   if (!strcmp(action, "add")){…
230
231  >   else if (!strcmp(action, "list")){…
258
259  >   else if (!strcmp(action, "read")){…
320
321  >   else if (!strcmp(action, "remove")){…
392
393  >   else if (!strcmp(action, "kill")){…
398
399      else{
400          close(server_fd);
401
402          exit(EXIT_SUCCESS);
403      }
404
405  }
```

Imagem 3: implementação, sem detalhes, de cada ação possível que pode ser enviada pelo cliente

É necessário remover o caractere newline da ação pois, no caso da ação kill, que não recebe nenhum parâmetro a mais, ela apresentará o caractere e não será possível realizar a comparação.

03 - Conclusão

O trabalho proposto conseguiu atingir suas metas, explicando como programas podem conversar entre si e como realizar isso na prática por meio da implementação de protocolos distintos. A linguagem C possui suas limitações, o que dificultou bastante a implementação tanto dos métodos de conexão entre servidor e cliente quanto do método de execução dos comandos descritos pelo cliente.

Como o servidor possui maior responsabilidade, foi necessário dedicar mais tempo para sua conclusão. O tempo foi dominado principalmente pela pesquisa e desenvolvimento de métodos de conexão e pela implementação das ações, que foram diretamente impactadas pelas limitações de estruturas de dados apresentadas pela linguagem C.