

INTRODUÇÃO

O trabalho visa mostrar como uma comunicação em redes pode ser feita através da comunicação entre sockets. Para tematizar o trabalho, foi simulada uma comunicação entre um client e um servidor onde o servidor é responsável por armazenar informações sobre sensores e equipamentos. O desenvolvimento do cliente e do servidor deve ser feito utilizando-se a linguagem C e a biblioteca padrão de C para sockets.

DESENVOLVIMENTO

Durante o desenvolvimento, utilizando como base o material disponibilizado no moodle, foi possível desenvolver um client e um servidor básico onde enviáramos mensagens do client para o servidor utilizando ipv4 e ipv6. Teríamos 4 tipos de comandos válidos que poderiam ser enviados pelo client - "add", "remove", "list", "read" - além de tratar mensagens inválidas fechando a conexão com o client. Para cada comando, deveríamos referenciar sensores e equipamentos e, por escolhas durante o desenvolvimento, manipular um array estático em C de 15 posições (limite de sensores instalados devido a especificação do trabalho) para simular um banco de dados.

DESAFIOS

Como desafios do trabalho, a manipulação de strings foi um empecilho, dado que exige uma manipulação em um nível mais baixo que em outras linguagens. O tratamento de erros do servidor também foi um desafio, dado que, além de ter que manipular strings, exigia uma atenção maior para os "casos de borda" como adicionar múltiplos sensores, remover múltiplos sensores, ler sensores que não estão instalados, etc. Também tive alguns problemas para tratar casos onde meu servidor e meu client se comunicavam com ipv4 ou ipv6, pois o tamanho das estruturas de dados para cada comunicação é diferente.

SOLUÇÕES IMPLEMENTADAS

Para o tratamento do tamanho das mensagens para diferentes protocolos, decidi retornar o tamanho da estrutura de dados como resultado da função que inicializa o client ou o servidor, exemplo:

```

int parse_client_address(const char *raw_address, const char *raw_port, struct sockaddr_storage *storage) {
    if (raw_address == NULL) return FALSE;
    int PORT_NUMBER = atoi(raw_port);
    if (PORT_NUMBER == 0) return FALSE;
    PORT_NUMBER = htons(PORT_NUMBER);
    struct in_addr ipv4_address;
    if (inet_pton(AF_INET, raw_address, &ipv4_address)) {
        struct sockaddr_in *ipv4_socket = (struct sockaddr_in *) storage;
        ipv4_socket->sin_port = PORT_NUMBER;
        ipv4_socket->sin_family = AF_INET;
        ipv4_socket->sin_addr = ipv4_address;
        return sizeof(struct sockaddr_in);
    }
    struct in6_addr ipv6_address;
    if (inet_pton(AF_INET6, raw_address, &ipv6_address)) {
        struct sockaddr_in6 *ipv6_socket = (struct sockaddr_in6 *) storage;
        ipv6_socket->sin6_port = PORT_NUMBER;
        ipv6_socket->sin6_family = AF_INET6;
        memcpy(&(ipv6_socket->sin6_addr), &ipv6_address, sizeof(ipv6_address));
        return sizeof(struct sockaddr_in6);
    }
    return FALSE;
}

```

Para o caso de manipular strings, usei principalmente os métodos `sprintf`, `strcat` e `strcpy`, como por exemplo:

```

char *get_list_success_response(int *sensors_in_equipment) {
    if (sensors_in_equipment == NULL) return "";
    static char response[BUFFER_SIZE_IN_BYTES] = "";
    memset(response, 0, BUFFER_SIZE_IN_BYTES);
    int *sensor;
    for (sensor = sensors_in_equipment; (int) *sensor != '\0'; sensor++) {
        char sensor_as_string[4] = " ";
        sprintf(sensor_as_string, "%d ", (int) *sensor);
        strcat(response, sensor_as_string);
    }
    strcat(response, "\n");
    return response;
}

```

CONSIDERAÇÕES FINAIS E FEEDBACK

Acredito que o trabalho ajudou a entender a comunicação através de sockets e tornar claro qual o papel de cada componente - cliente e servidor. Outro ponto é que, por não ser o objetivo do trabalho, não foi necessário implementar uma comunicação com banco de dados - o que poderia ser uma boa ideia dado que a comunicação também pode ser feita em rede - então utilizamos apenas arrays em memória para a implementação, porém no mercado essa não é uma solução viável. Acredito que um complicador pode ter sido a linguagem escolhida, se o objetivo do trabalho for unicamente mostrar aos alunos como a

comunicação com sockets funciona, considerar o uso de outras linguagens afim de facilitar o aprendizado pode ser uma boa ideia.