UNIVERSIDADE FEDERAL DE SANTA MARIA

Nome: Carla de Oliveira Barden **Matrícula:** 201713228

Simulador de Casa Inteligente

Uma casa inteligente controla automaticamente a temperatura e a iluminação de seus ambientes. Ela também é capaz de abrir e fechar as portas, janelas e cortinas através de comandos, sendo que a porta de entrada usa biometria para permitir o acesso ao interior da casa.

Logo, será necessário um software simulador, com tarefas periódicas para a simulação da leitura dos sensores (temperatura, presença e biometria) e um software de controle para o dono da casa regular sua preferência de temperatura e intensidade da luz, assim como abrir e fechar as portas, janelas e cortinas assim que ele desejar. Ambos os softwares se comunicarão via rede.

Neste trabalho, será simulada uma **sala inteligente**, conforme o seguinte:

- Uma porta principal, com sensor biométrico;
  Ela também poderá ser aberta e fechada através de comandos.
- Luz inteligente, através do sensor de presença e horário pré-programado;
  Sua intensidade poderá ser regulada através de comandos, assim como a sua ativação e desligamento.
- Temperatura inteligente, conforme leitura do sensor e das preferências predefinidas do usuário;
  A temperatura também poderá ser ajustada através de comandos.
- Uma janela com cortinas, que também poderão ser abertas e fechadas através de comandos do usuário.

Código

Do trabalho proposto, estão prontas as comunicações, o protocolo a ser usado e um pré-validador de comandos enviados pela rede, funcionando, basicamente, da seguinte maneira:

- Os fontes estão nas pastas *src/cli* e *src/srv*, que representam, respectivamente, os códigos necessários ao cliente e ao servidor;
- O servidor, neste caso, representa o *monitor*, isto é, o software de controle;
- O cliente, neste caso, representa o *simulador*, ou seja, o software que simula os embarcados;
- Há, também, um *makefile*, portanto:
    - Para compilar somente o software servidor: $ *make servidor*
    - Para compilar somente o software cliente: $ *make cliente*
    - Para compilar ambos os softwares: $ *make*
    - Para executar o software servidor *: $ *make executar_servidor*
    - Para executar o software cliente*: $ *make executar_cliente*
    - Para limpar somente o software servidor: $ *make limpar_servidor*
    - Para limpar somente o software cliente: $ *make limpar_cliente*
    - Para limpar tudo: $ *make clean*

    ** Iniciar cada software em um terminal. Recomendável iniciar o servidor antes do cliente.*
- Após compilar (se usado o *make*), serão geradas as pastas *obj/* e *bin/*, contendo os arquivos-objeto e os executáveis.

Ambos os softwares enviam e recebem dados via rede (sockets). No software *monitor*, é possível digitar comandos (conforme o protocolo) para enviá-los ao software *simulador*, o qual os receberá e enviará um feedback para o *monitor* (que o imprimirá na tela). Já o software *simulador* lê os dados da rede e executa um *parser* sobre eles, para obter os comandos conforme as necessidades impostas pelo protocolo implementado, os imprime na tela e envia um feedback para o software *monitor*. **Não é possível digitar comandos no software *simulador*. Apenas o software *monitor* os aceita.** Os dois softwares possuem threads diferentes para a leitura e a escrita na rede. No caso da escrita, é utilizada uma

variável de condição para efetuá-la somente quando necessário, já que esta é uma operação bloqueante e o não uso de variáveis de condição acarretaria em espera ocupada.

O protocolo é dado pela estrutura:

```
struct __prot__{
        char comando[10];
        char alvo[10];
        char opt[4];
        char valor[3];
};
```

Aqui está um exemplo de um comando, da forma *comando alvo opt valor* (os comandos em si ainda não foram implementados):

ajustar temp add 2

Ele será usado para aumentar 2ºC (somar 2) à temperatura atual. Nem todos os comandos terão *alvo, opt* e *valor*. Por enquanto, a validação que está sendo feita diz respeito ao tamanho de cada argumento do comando, sendo aceitos vetores de caracteres de tamanho *[n-1]* para cada item da estrutura, já que deve-se garantir a existência do *"\0"* no final da string. O comando *sair* já está funcionando. Há, ainda, alguns *warnings* não devidamente tratados.

PS: Se utilizado o *make* para executar, pode ser que, ao se tentar executar 2x consecutivas, ele acuse erro ao aceitar a conexão, por motivos ainda desconhecidos (timeout do tcp/ip, talvez). Basta esperar um pouco e tentar novamente que ele funcionará.