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;
  Sua intensidade poderá ser regulada através de comandos, assim como a sua ativação e desligamento.
- Ar condicionado e sensor de temperatura;
  A temperatura do ar também poderá ser ajustada através de comandos.
- Uma janela com cortinas, que também poderão ser abertas e fechadas através de comandos do usuário.

Código

Os códigos fontes do trabalho proposto estão organizados da seguinte maneira:

- Os fontes estão nas pastas *src/cli*, *src/shr* e *src/srv*, que representam, respectivamente, os códigos necessários ao cliente, os códigos compartilhados entre cliente e servidor e os códigos necessários 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 quadro abaixo) 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, executa o comando (se possível) 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.

Comandos Implementados				
Comando	**Alvo**	**Opt**	**Valor**	**Descrição**
abrir	*porta*			Abre o alvo especificado
abrir	*janela*			
abrir	*cortina*			
fechar	*porta*			Fecha o alvo especificado
fechar	*janela*			
fechar	*cortina*			
ligar	*ar*			Liga o alvo especificado
ligar	*luz*			
desligar	*ar*			Desliga o alvo especificado
desligar	*luz*			
ajustar	*luz*	*mea*		Luz com 50% de intensidade / cortina entreaberta
ajustar	*cortina*	*mea*		
ajustar	*temp*	*set*	*[valor inteiro]*	Ajusta a temperatura para o valor desejado (entre 17 e 24).
ajustar	*temp*	*add*	*[valor inteiro]*	Soma *[valor inteiro]* da temperatura atual (valor da temperatura entre 17 e 24)
ajustar	*temp*	*dim*	*[valor inteiro]*	Diminui *[valor inteiro]* da temperatura atual (valor da temperatura entre 17 e 24)
ver	*porta*			Retorna o estado do alvo especificado.
ver	*janela*			
ver	*cortina*			
ver	*luz*			

Comando	Alvo	Opt	Valor	Descrição
ver	*temp*			Retorna temperatura caso o ar condicionado estiver ligado
ver	*term*			Retorna indicação do termômetro (leitura do sensor)
cadbio	*[valor inteiro]*			Cadastro de "biometria" (simbólico)
auto	*luz*			Retorna ao controle da iluminação via sensor de presença
reset				Reinicia sistema para valores padrão
sair				Encerra a aplicação

O protocolo é dado pela estrutura:

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

Nem todos os comandos ocupam todos os campos da estrutura. A validação do protocolo/comandos que está sendo feita no lado do monitor (servidor) 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. No lado do simulador (cliente), é realizada a verificação. Se não for um dos comandos descritos no quadro acima, o software exibirá a mensagem "*Comando Inválido!"*. **Não se deve digitar espaços antes ou depois do comando.** Os campos do comando são separados por espaços.

Conforme dito anteriormente, no software *monitor* (servidor) é possível digitar comandos, pré validados através do seu formato e tamanho, para enviá-los, via rede (sockets), para a execução no software simulador. Também recebe-se o retorno (feedback) do comando (se foi possível executá-lo ou se o comando é inválido).

Já o software *simulador* (cliente) é um pouco mais complexo. Além de verificar se o comando é válido e executá-lo, o software simulador também conta com três tarefas periódicas, executando em threads distintas, sendo elas:

- *ler_presenca:* efetua a "leitura" de um "sensor" de presença simulado através da função *ler_sensor_presenca* (que gera valores pseudoaleatórios para emular a leitura). Caso o sensor "constate" presença, acenderá a luz automaticamente. O período desta tarefa é de 1 segundo.
- *ler_biometria:* efetua a "leitura" de um "sensor" de biometria simulado e compara com a "biometria" (valor inteiro, o padrão é 1234, mas pode ser modificado através do comando *cadbio)*. Caso sejam compatíveis, automaticamente, a tarefa abre a porta, espera três segundos e fecha a porta. Esta "leitura" é gerada pseudoaleatoriamente pela função *ler_sensor_biometria.* O período desta tarefa é 1 segundo.
- *ler_temperatura:* efetua a "leitura" de um "sensor" de temperatura simulado pela função *ler_sensor_temperatura* e compara com a temperatura ajustada pelo usuário caso o ar esteja ligado. Caso leia um valor diferente, corrige a temperatura automaticamente. Novamente, a "leitura" é gerada pseudoaleatoriamente. O período desta tarefa é de 5 segundos.

No terminal do software simulador é possível verificar os valores que estão sendo lidos pelos sensores, além dos comandos recebidos via rede e quando ocorre alguma ação automática (acender a luz quando o sensor de presença detecta alguém, por exemplo).

Há, também, uma thread responsável por executar os comandos enviados pelo usuário. Ela também só é ativada quando necessário, via variável de condição, para evitar espera ocupada. No caso específico da luz, o programa não interferirá na iluminação caso o usuário a ajuste, independente da indicação do sensor de presença. Para devolver o controle da iluminação ao programa, basta usar o comando *auto luz*.

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.