Trabalho Prático 2 -
Redes de Sensores Sem Fio

Henrique da Fonseca Diniz Freitas
[**Matrícula:** 2021031688]

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte - MG - Brasil

henriquefdf@ufmg.br

1 Introdução

Este projeto consiste na implementação de um **sistema de Redes de Sensores Sem Fio (RSSF) simplificado**, utilizando **sockets TCP** e **threads** na linguagem C (interface POSIX). Cada sensor é um *cliente* que se conecta a um *servidor* central, e as mensagens trocadas seguem um modelo de *publish/subscribe*, em que **sensores de um mesmo tipo** (por exemplo, `temperature`) recebem e atualizam suas medições uns dos outros.

Os objetivos principais incluem:

- Criar um **servidor** que aceite conexões simultâneas via múltiplas threads.
- Criar um **cliente** (sensor) que:
  - Lê parâmetros de linha de comando (endereço do servidor, porta, tipo e coordenadas).
  - Envia suas medições periodicamente (com intervalos baseados no tipo de sensor).
  - Recebe e processa medições de outros sensores do mesmo tipo.
  - Gerencia uma lista de até 3 **vizinhos mais próximos** e atualiza seu valor com base na distância e diferença de medição, segundo a fórmula definida.
- Exibir **logs** padronizados tanto no servidor quanto nos clientes.
- Tratar **desconexões** de clientes, enviando e recebendo medições `-1.0000` para removê-los do sistema.

2 Arquitetura e Organização

A solução foi dividida em dois programas:

2.1 Servidor (`server.c`)

- Utiliza `socket()`, `bind()`, `listen()` para aceitar conexões indefinidamente.
- Cria uma **thread** (`pthread_create()`) para cada cliente conectado, responsável por receber e encaminhar mensagens.
- Mantém, em uma estrutura global, os dados (tipo, coordenadas, socket) de todos os clientes conectados.
- Encaminha mensagens de um sensor para todos os sensores do mesmo tipo.
- Detecta **desconexões** (via `recv()`) e:
  - Envia uma mensagem de valor `-1.0000` para os sensores do mesmo tipo.
  - Remove o cliente desconectado da lista global.

2.2 Cliente (`client.c`)

- Valida parâmetros recebidos: endereço do servidor, porta, tipo e coordenadas (0–9).
- Conecta ao servidor e:
  - Gera uma medição inicial aleatória dentro dos limites do tipo (e.g., 20.0–40.0 para `temperature`).
  - Envia medições periodicamente (a cada 5s, 7s ou 10s, dependendo do tipo).
  - Recebe mensagens retransmitidas pelo servidor e:
    - Ignora mensagens de sensores na mesma coordenada (`action: same location`).
    - Remove sensores que desconectaram (`action: removed`).
    - Aplica a fórmula de correção se o remetente for um dos 3 mais próximos; caso contrário, descarta com `action: not neighbor`.

3 Funcionamento e Fluxo de Dados

3.1 Estrutura da Mensagem

Cada mensagem segue o formato:

```
struct sensor_message {
    char  type[12];    // "temperature", "humidity", "air_quality"
    int   coords[2];   // Ex.: (2,3)
    float measurement; // Valor do sensor, ex.: 25.3467
};
```

3.2 Correção de Medição

A fórmula para atualizar o valor de um sensor é:

$$\text{nova\_medição} = \text{medição\_atual} + 0.1 \times \frac{1}{(d+1)} \left(\text{medição\_remota} - \text{medição\_atual}\right)$$

onde $d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. Valores fora dos limites são ajustados (*clamp*).

3.3 Gerenciamento de Vizinhos

Os sensores mantêm uma lista de sensores conhecidos, mas apenas corrigem suas medições com base nos 3 vizinhos mais próximos, recalculados dinamicamente a cada mensagem recebida.

3.4 Tratamento de Desconexões

Quando um sensor desconecta (Ctrl + C), o servidor avisa os sensores do mesmo tipo com uma mensagem `-1.0000`, que eles interpretam como `action: removed`.

4 Logs

4.1 Servidor

Ao receber uma mensagem:

```
log:
<type> sensor in (<x>,<y>)
measurement: <measurement>
```

Desconexões são exibidas com `measurement: -1.0000`.

4.2 Cliente

Ao receber uma mensagem:

```
log:
<type> sensor in (<x>,<y>)
measurement: <measurement>
action: <same location | not neighbor | removed | correction of X>
```

5 Testes Realizados

Testes foram realizados para validar todas as funcionalidades. Sensores próximos corrigem mutuamente suas medições; desconexões são registradas e tratadas com `removed`; sensores na mesma célula são ignorados. Limites foram respeitados (`clamp`), e diferentes tipos de sensores operaram em paralelo sem interferência.

6 Desafios e Soluções

6.1 Uso de Threads no Servidor

Desafio: Garantir que múltiplas threads acessando a lista global de clientes não causassem *race conditions*. **Solução:** Uso de **mutexes** (`pthread_mutex_lock`/`unlock`) para sincronizar inserção, remoção e modificações na lista.

6.2 Lógica dos Vizinhos

Desafio: Implementar uma lógica eficiente para determinar os 3 vizinhos mais próximos e tratar sensores em coordenadas iguais. **Solução:** Criação de uma função para calcular distâncias e retornar os índices dos 3 menores valores. Sensores na mesma célula são ignorados para correção (`same location`).

6.3 Outros Imprevistos

- **Mensagens fora de ordem**: Um sensor novo podia não ser reconhecido imediatamente como vizinho. A solução foi armazenar todas as mensagens e recalcular vizinhos dinamicamente.
- **Ponto flutuante**: Para evitar erros ao comparar `measurement` com `-1.0000`, usamos uma tolerância (`fabsf(valor + 1.0f) < 0.0001f`).

7 Conclusão

O projeto demonstrou como modelar um sistema básico de **Redes de Sensores Sem Fio (RSSF)** com *Sockets TCP* e **Threads**, integrando conceitos de concorrência, modelo *publish/subscribe*, e gerenciamento dinâmico de vizinhos. Os principais desafios (concorrência e vizinhança) foram resolvidos com sucesso, resultando em um sistema robusto e funcional.