Trabalho Prático 2 - Redes de Sensores Sem Fio
Entrega Individual

Data de Entrega: 08 de janeiro de 2025, as 23:59

Introdução

As Redes de Sensores Sem Fio (RSSF) representam uma tecnologia crucial para a coleta de dados em tempo real em ambientes diversos, com aplicações como monitoramento ambiental, automação industrial e até atuações dentro de seres vivos. Essas redes consistem em pequenos dispositivos sensores, geralmente distribuídos em uma área, capazes de captar informações como temperatura, umidade, pressão, etc. Cada sensor se comunica sem fio com outros dispositivos ou diretamente com um servidor central, permitindo a criação de sistemas distribuídos e escaláveis. Nas figuras abaixo, um exemplo de um diagrama de rede e um sensor, ambos para situações embaixo d'água:

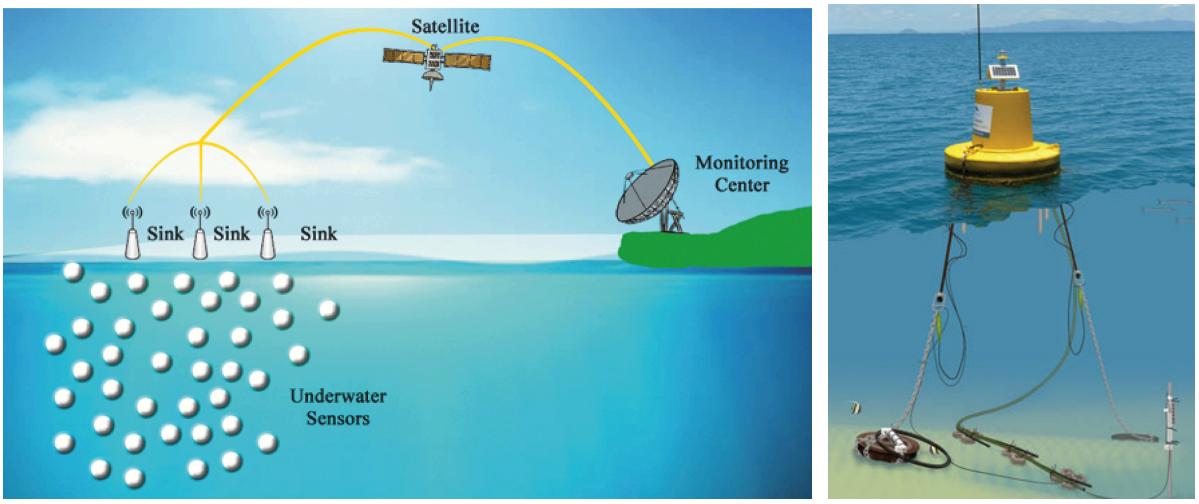

Imagem 1 - Sala de Espera do jogo *Among Us*

Para ilustrar o funcionamento de uma RSSF, este trabalho prático tem como objetivo simular uma rede composta por três tipos de sensores espalhados em uma região delimitada. Os sensores serão responsáveis por coletar e reportar informações de forma periódica. Esses sensores se comunicarão com um servidor central utilizando um modelo *publish/subscribe*, onde as mensagens serão enviadas a tópicos específicos de acordo com o tipo de sensor. Clientes conectados a esses tópicos poderão receber os dados e atualizar suas medições com base na proximidade aos sensores.

Objetivo

Você desenvolverá 2 programas para o sistema de salas de espera utilizando apenas as funcionalidades da biblioteca de sockets POSIX e a comunicação via protocolo TCP. O código do servidor **deverá utilizar múltiplas threads** para a manutenção das múltiplas conexões. As próximas seções detalham o que cada entidade (servidor e cliente) deve fazer:

Os objetivos deste trabalho são:

- Desenvolver um servidor utilizando a interface de sockets na linguagem C;
- Desenvolver um cliente utilizando a interface de sockets na linguagem C;
- Permitir que múltiplos clientes sejam conectados ao servidor, e que ele encaminhe as mensagens dos sensores;
- Fazer com que os clientes reportem as medições de tempos em tempos, e que atualizem suas medidas com base nos vizinhos.

Protocolo

O protocolo de comunicação entre o cliente e o servidor será modelado de tal forma que as operações sejam representadas pela seguinte estrutura:

```
struct sensor_message {
     char[12] type;
     pair<int,int> coords;
     float measurement;
};
```

Abaixo descrição de cada uma das propriedades:

type
Essa propriedade armazena o tipo do sensor. Neste trabalho serão 3 diferentes:

- `temperature`, que vai medir a temperatura do ambiente;
- `humidity`, para sensores de umidade;
- `air_quality`, para medir a qualidade do ar;

coords
Essa propriedade representa as coordenadas do sensor no espaço. Neste trabalho será considerado um grid 10x10, onde o primeiro valor do par representa em que “linha” o sensor está, e o segundo a “coluna”.

measurement
Já essa propriedade representa a medição de cada sensor, que ele mantém e envia em cada intervalo de tempo, além de atualizar conforme os vizinhos.

Inicialização

O código do cliente deve ser executado passando como argumento as coordenadas. Conforme mencionado anteriormente, o espaço dos sensores é um grid 10x10, de forma que as coordenadas em cada eixo vão variar de 0 à 9, e devem ser passadas da seguinte forma:

```
./client 127.0.0.1 51511 -type <temperature|humidity|air_quality>
                         -coords <x> <y>
```

O usuário pode passar parâmetros errados na hora de inicializar o cliente, onde a ordem prioritária dos erros, bem como as mensagens que devem ser exibidas, são:

Tabela 1 - Erros de inicialização e tratativas

Descrição do Erro	Exemplo	Mensagem
Faltando argumentos	`./client 127.0.0.1 51511 -type temperature -coords 5`	Error: Invalid number of arguments
Argumento -type faltando	`./client 127.0.0.1 51511 -tipo temperature -coords 5 7`	Error: Expected '-type' argument
Tipo de sensor não definido	`./client 127.0.0.1 51511 -type pression -coords 5 7`	Error: Invalid sensor type
Argumento -coords faltando	`./client 127.0.0.1 51511 -type temperature -pos 5 7`	Error: Expected '-coords' argument
Coordenadas fora do intervalo	`./client 127.0.0.1 51511 -type temperature -coords 10 7`	Error: Coordinates must be in the range 0-9

OBS: Após cada mensagem de erro, deve-se exibir também a seguinte mensagem padrão na linha de baixo:

Usage: `./client <server_ip> <port> -type <temperature|humidity|air_quality> -coords <x> <y>`

Também, erros que não estão listados (como passar chars ao invés de inteiros) não precisam ser tratados, e não serão testados na avaliação.

Exemplo completo de erro e mensagem:

```
> ./client 127.0.0.1 51511 -type temperature -coords 5
Error: Invalid number of arguments
Usage: ./client <server_ip> <port> -type <tipos> -coords <x> <y>
```

, onde `<tipos>` = `<temperature|humidity|air_quality>` (só para destacar que é na mesma linha).

Uma vez corretamente inicializado, o sensor vai realizar sua medição, que consiste em um valor aleatório no intervalo descrito na Tabela 2. A cada intervalo de tempo também descrito nesta tabela, o cliente vai mandar uma mensagem para o servidor, informando sua localização, tipo e medição.

Tabela 2 - Valores e Intervalos dos Sensores

Tipo do sensor	**Intervalo das medições**	**Intervalo entre mensagens**
Temperatura (ºC)	20.0 - 40.0	5 segundos
Humidade (%)	10.0 - 90.0	7 segundos
Qualidade do ar (μg/m³)	15.0 - 30.0	10 segundos

Funcionamento da Rede

Após a inicialização dos valores, o funcionamento do sistema segue uma abordagem baseada no padrão *publish/subscribe*. Nesse sistema, cada sensor **já é** inscrito no tópico correspondente ao seu tipo (por exemplo, *temperature*, *humidity* ou *air_quality*), de forma que todas as mensagens enviadas pelos sensores de um determinado tipo são transmitidas para todos os sensores do mesmo tipo. O servidor é responsável por gerenciar os tópicos, garantindo que as mensagens sejam encaminhadas corretamente para os sensores interessados.

Os sensores enviam suas medições periodicamente para o tópico correspondente, e o servidor as propaga para todos os sensores inscritos. Cada sensor, ao receber uma nova medição de outro sensor, utiliza essa informação para corrigir sua própria medição com base na proximidade e na diferença entre os valores, caso seja um dos três sensores mais próximos. Isso significa que somente a primeira medição (quanto o cliente é inicializado) que escolhida aleatoriamente, o restante são só ajustes nessa escolha.

A fórmula para atualizar a medição é dada da seguinte forma:

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

Onde:

- *nova_medição*: é o novo valor do sensor após a correção.
- *medição_atual*: é o valor atual do sensor.
- *medição_remota*: é o valor enviado pelo sensor remoto.
- *d*: é a distância euclidiana entre o sensor atual e o sensor remoto:

  $$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Essa fórmula leva em conta tanto a diferença entre o valor do sensor que recebeu a mensagem e o valor atual quanto a proximidade entre os sensores. Se após a atualização o valor corrigido ultrapassar o intervalo permitido (Tabela 2), ele deve ser ajustado automaticamente para o limite mais próximo (inferior ou superior). Isso assegura que todas as medições permaneçam válidas dentro do contexto da aplicação.

Conforme já citado, um aspecto crucial do sistema é o gerenciamento dos vizinhos mais próximos. Cada sensor deve manter a informação de seus vizinhos, pois **vai atualizar sua medição somente com base nos três sensores mais próximos** de sua posição, descartando as medidas dos outros. Manter todos os vizinhos, e não só os 3 mais próximos, é importante porque caso um dos sensores próximos saia do sistema, o sensor atual deve identificar o "próximo da fila" para substituí-lo na “lista de vizinhos mais próximos”. Essa feature é importante considerando as RSSF, que têm que lidar constantemente com sensores saindo da rede (e.g. sem bateria, destruído pelo ambiente, etc.).

Descoberta Dinâmica de Sensores

A descoberta de novos sensores no sistema é dinâmica. Quando um novo sensor entra na rede, ele pode começar recebendo (e consequentemente utilizando para atualizar a sua) medições de sensores que não necessariamente são seus vizinhos mais próximos (considerando todos da rede), devido à falta de conhecimento inicial sobre o ambiente. Conforme novas mensagens são recebidas, o sensor atual ajusta sua lista de vizinhos e passa a priorizar as medições dos mais próximos.

Para exemplificar, considere o seguinte cenário: um ambiente com 5 sensores do mesmo tipo e um novo sensor desse mesmo tipo entra no sistema:

- Se ele receber mensagens na ordem do mais distante para o mais próximo, ele realizará **5 atualizações** até estabilizar com os três vizinhos mais próximos.
- Se a ordem for do mais próximo para o mais distante, apenas **3 atualizações** serão realizadas, pois ele estabiliza sua lista de vizinhos antes de processar as medições mais distantes.

Esse comportamento dinâmico ilustra o procedimento real de muitos sistemas em RSSF para descobrir seus vizinhos, e a fórmula de atualização também considera esse caso, ao considerar a distância e, consequentemente, diminuir a influência de sensores distantes.

Exibição das mensagens

O servidor deve exibir todas as mensagens recebidas em um formato padronizado, permitindo fácil rastreamento e depuração:

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

, onde:

- `<type>`: Tipo do sensor (temperature, humidity ou air_quality).
- `(<x>,<y>)`: Coordenadas do sensor que enviou a mensagem.
- `<measurement>`: Valor da medição enviado.

Os clientes, por sua vez, vão exibir todas as mensagens que chegam para ele, ou seja, todas do mesmo tipo dele. Isso inclui a deles mesmos, já que não devem exibir nada ao enviar uma mensagem. O formato de exibição é o mesmo do servidor, mas com um campo a mais:

```
log:
<type> sensor in (<x>,<y>)
measurement: <measurement>
action: <status>
<\n>
```

Esse último campo descreve o tratamento realizado daquela mensagem, devendo ser exibido:

- `not neighbor`: Caso o sensor que recebeu considere que o emissor está muito longe para ser considerado vizinho, e descarta a medição; Essa ação

só é possível caso o receptor já tenha três vizinhos, e essa nova mensagem venha de um mais distante que esses três.
- `correction of <value>`: Caso onde o sensor é um vizinho, e portanto é feita correção na medição. O valor a ser exibido é o que vai ser corrigido da medição atual, arredondado para **quatro casas decimais**. Se negativo, indicar com sinal "-" na frente;
- `same location`: Caso o sensor emissor esteja na mesma célula que o receptor, e portanto não deve ser considerada (não é vizinho);
- `removed`: Caso em que o sensor saiu do sistema (explicado no tópico abaixo);

Perceba que é possível sensores ocuparem o mesmo espaço do grid (não bloqueamos isso na inicialização), e como não passamos nenhum ID de cada sensor, não temos como saber se a mensagem que chegou é a que o próprio sensor enviou, ou de outro. Assim, para simplificar, vamos considerar como não vizinho e descartar todas as medições da mesma célula que o sensor.

Obs: Existe um <\n> no final do log, que significa para adicionar mais uma quebra de linha, para separar melhor os logs no terminal, dando uma linha em branco entre eles.

Encerramento de um sensor

Os clientes não vão receber nenhum input por parte do usuário. Isso é proposital, já que em Redes de Sensores Sem Fio os dispositivos são autônomos e descartáveis. Assim, para finalizar o sensor é necessário parar o processo do cliente (Ctril + C), o que representa a destruição/inatividade do dispositivo. O servidor, por sua vez, vai perceber que o cliente foi desconectado, e deve exibir o log padrão, mas com medida negativa (-1.0000), e enviar essa medição para os sensores do mesmo tipo como se fosse uma medição normal desse sensor, só que com valor negativo.

```
log:
<type> sensor in (<x>,<y>)
measurement: -1.0000
```

Os clientes, por sua vez, devem interpretar que essa medição negativa representa que esse sensor não está mais disponível, e devem removê-lo de suas bases de dados. Devem exibir a *action* como *removed*.

Exemplo de Execução

Esta seção apresenta alguns exemplos de execuções do sistema. A fim de facilitar a

explicação, as tabelas a seguir detalham o passo a passo das mensagens ao longo do tempo. Nesse primeiro exemplo temos dois sensores do mesmo tipo se conectando no servidor inicialmente sem sensores. Considere que o Sensor 1 teve medição inicial de 25 ºC e o segundo 20 ºC, e que o envio e recebimento de mensagens é instantâneo.

Tempo (1 s)	**Servidor**	**Terminal 1**	**Terminal 2**
T0		`> ./client 127.0.0.1 51511 -type temperature -coords 2 2`	
T1			
T2			`> ./client 127.0.0.1 51511 -type temperature -coords 4 4`
T3			
T4			

T5	`log:`
	`temperature sensor in (2,2)`
	`measurement: 25.0000`
		`log:`
		`temperature sensor in (2,2)`
		`measurement: 25.0000`
		`action: same location`
			`log:`
			`temperature sensor in (2,2)`
			`measurement: 25.0000`
			`action: correction of 0.1306`

T6			

T7	`log:`
	`temperature sensor in (4,4)`
	`measurement: 20.1306`
		`log:`
		`temperature sensor in (4,4)`
		`measurement: 20.1306`
		`action: correction of −0.1272`
			`log:`
			`temperature sensor in (4,4)`
			`measurement: 20.1306`
			`action: same location`

T8			
T9			

T10	`log:`
	`temperature sensor in (2,2)`
	`measurement: 24.8728`
		`log:`
		`temperature sensor in (2,2)`
		`measurement: 24.8728`
			`log:`
			`temperature sensor in (2,2)`
			`measurement: 24.8728`

		action: same location	action: correction of 0.1239
T11			
T12	log: temperature sensor in (4,4) measurement: 20.2542	log: temperature sensor in (4,4) measurement: 20.2545 action: correction of −0.1206	log: temperature sensor in (4,4) measurement: 20.2545 action: same location

Abaixo, um exemplo com 3 sensores, um de cada tipo, e entra um quarto sensor. Considere as medidas iniciais como:

- **Sensor 1**: Tipo *temperature*, posição (2,2), medição inicial 25.0;
- **Sensor 2**: Tipo *humidity*, posição (3,3), medição inicial 50.0;
- **Sensor 3**: Tipo *air_quality*, posição (1,1), medição inicial 22.0;
- **Sensor 4**: Tipo *temperature*, posição (4,4), medição inicial 20.0;

Tempo	Servidor	Terminal 1	Terminal 2	Terminal 3	Terminal 4
T0		> ./client 127.0.0.1 51511 -type temperature -coords 2 2	> ./client 127.0.0.1 51511 -type humidity -coords 3 3	> ./client 127.0.0.1 51511 -type air_quality -coords 1 1	
T1					
T2					
T3					> ./client 127.0.0.1 51511 -type temperature -coords 4 4
T4					
T5	log: temperature sensor in (2,2)	log: temperature sensor in (2,2)			log: temperature sensor in (2,2)

	measurement: 25.0000	measurement: 25.0000 action: same location			measurement: 25.0000 action: correction of 0.1306
T6					
T7	log: humidity sensor in (3,3) measurement: 50.0000		log: humidity sensor in (3,3) measurement: 50.0000 action: same location		
T8	log: temperature sensor in (4,4) measurement: 20.1306	log: temperature sensor in (4,4) measurement: 20.1306 action: correction of −0.1272			log: temperature sensor in (4,4) measurement: 20.1306 action: same location
T9					
T10	log: temperature sensor in (2,2) measurement: 24.8728 log: air_quality sensor in (1,1) measurement: 22.0000	log: temperature sensor in (2,2) measurement: 24.8728 action: same location		log: air_quality sensor in (1,1) measurement: 22.0000 action: same location	log: temperature sensor in (2,2) measurement: 24.8728 action: correction of 0.1239
T11					
T12					

T13	log: temperature sensor in (4,4) measurement: 20.2545	log: temperature sensor in (4,4) measurement: 20.2545 action: correction of −0.1206			log: temperature sensor in (4,4) measurement: 20.2545 action: same location
T14	log: humidity sensor in (3,3) measurement: 50.0000		log: humidity sensor in (3,3) measurement: 50.0000 action: same location		

Por fim, um exemplo em que um sensor se desconecta. Considere os seguintes sensores:

- **Sensor 1**: Tipo *humidity*, posição (2,2), medição atual de 50.0;
- **Sensor 2**: Tipo *humidity*, posição (2,3), medição atual de 55.0;
- **Sensor 3**: Tipo *humidity*, posição (3,2), medição atual de 58.0;
- **Sensor 4**: Tipo *humidity*, posição (3,3), medição atual de 52.0;
- **Sensor 5**: Tipo *humidity*, posição (9,9), medição atual de 60.0;

Cada um entrou no sistema com um segundo de diferença, em ordem crescente (ou seja, primeiro o sensor 1 vai mandar mensagem, depois o 2, etc.). Nesse exemplo os sensores já trocaram algumas mensagens, de forma que os dispositivos 1, 2 e 3 e 4 são vizinhos entre si. Contudo, quando o sensor 2 sai do sistema, os outros consideram o 5 como um vizinho próximo:

Tempo	Terminal 1	Terminal 2	Terminal 3	Terminal 4	Terminal 5
T0	log: humidity sensor in (2,2) measurement: 50.0000 action: same location	log: humidity sensor in (2,2) measurement: 50.0000 action: correction of −0.2500	log: humidity sensor in (2,2) measurement: 50.0000 action: correction of 0.1000	log: humidity sensor in (2,2) measurement: 50.0000 correction of −0.0828	log: humidity sensor in (2,2) measurement: 50.0000 action: not neighbor
T1	log: humidity sensor in (2,3) measurement: 54.7500 action: correction of 0.2375	log: humidity sensor in (2,3) measurement: 54.7500 action: action: same location	log: humidity sensor in (2,3) measurement: 54.7500 action: correction of 0.2755	log: humidity sensor in (2,3) measurement: 54.7500 action: correction of 0.1416	log: humidity sensor in (2,3) measurement: 54.7500 action: correction of -0.0514
T2	log: humidity sensor in (3,2) measurement: 48.3755 action: correction of -0.0931	log: humidity sensor in (3,2) measurement: 48.3755 action: correction of -0.2640	log: humidity sensor in (3,2) measurement: 48.3755 action: same location	log: humidity sensor in (3,2) measurement: 48.3755 action: correction of -0.1842	log: humidity sensor in (3,2) measurement: 48.3755 action: correction of -0.1132

T3	log: humidity sensor in (3,3) measurement: 51.8746 action: correction of 0.0717	log: humidity sensor in (3,3) measurement: 51.8746 action: correction of -0.1306	log: humidity sensor in (3,3) measurement: 51.8746 action: correction of 0.1750	log: humidity sensor in (3,3) measurement: 51.8746 action: same location	log: humidity sensor in (3,3) measurement: 51.8746 action: correction of -0.0839
T4	log: humidity sensor in (9,9) measurement: 59.7515 action: not neighbor	log: humidity sensor in (9,9) measurement: 59.7515 action: not neighbor	log: humidity sensor in (9,9) measurement: 59.7515 action: not neighbor	log: humidity sensor in (9,9) measurement: 59.7515 action: not neighbor	log: humidity sensor in (9,9) measurement: 59.7515 action: same location
T5					
T6	log: humidity sensor in (2,3) measurement: -1.0000 action: removed	Ctril + C (- Cliente se desconecta - Servidor manda mensagem com medida -1.0000)	log: humidity sensor in (2,3) measurement: -1.0000 action: removed	log: humidity sensor in (2,3) measurement: -1.0000 action: removed	log: humidity sensor in (2,3) measurement: -1.0000 action: removed

T7	log: humidity sensor in (2,2) measurement: 50.2161 action: same location		log: humidity sensor in (2,2) measurement: 50.2161 action: correction of 0.0833	log: humidity sensor in (2,2) measurement: 50.2161 action: correction of -0.0687	log: humidity sensor in (2,2) measurement: 50.2161 action: correction of -0.0875
T8					
T9	log: humidity sensor in (3,2) measurement: 48.6337 action: correction of -0.0791		log: humidity sensor in (3,2) measurement: 48.6337 action: same location	log: humidity sensor in (3,2) measurement: 48.6337 action: correction of -0.1586	log: humidity sensor in (3,2) measurement: 48.6337 action: correction of -0.1079
T10	log: humidity sensor in (3,3) measurement: 51.6473 action: correction of 0.0626		log: humidity sensor in (3,3) measurement: 51.6473 action: correction of 0.1507	log: humidity sensor in (3,3) measurement: 51.6473 action: same location	log: humidity sensor in (3,3) measurement: 51.6473 action: correction of -0.0834

T11	`log: humidity sensor in (9,9) measurement: 59.4727 action: correction of 0.0851`		`log: humidity sensor in (9,9) measurement: 59.4727 action: correction of 0.1046`	`log: humidity sensor in (9,9) measurement: 59.4727 action: correction of 0.0825`	`log: humidity sensor in (9,9) measurement: 59.4727 action: same location`

Detalhes de implementação

Gerenciamento de Sensores:

- O servidor será responsável por manter o registro de todos os sensores conectados, incluindo suas coordenadas e tipo.
- Os sensores devem manter a lista dos vizinhos mais próximos. Caso um vizinho "caia", o sensor deve atualizar sua lista, promovendo o "próximo da fila" à posição de vizinho próxmo.

Tipos de Sensores:

- Apenas três tipos de sensores serão aceitos: `temperature`, `humidity` e `air_quality`. Qualquer tipo diferente deve resultar em um erro e impedir a execução do cliente.

Coordenadas dos Sensores:

- As coordenadas dos sensores devem ser especificadas ao iniciar o cliente e precisam estar dentro do intervalo [0, 9].
- Caso as coordenadas estejam fora do intervalo ou não sejam numéricas, o cliente deve exibir uma mensagem de erro e não iniciar.

Mensagens e Tópicos:

- Cada tipo de sensor possui um tópico exclusivo, no qual serão publicadas e recebidas as mensagens.
- Mensagens de sensores do mesmo tipo devem ser encaminhadas apenas para outros sensores do mesmo tipo.

Atualização de Medições:

- As medições iniciais são aleatórias dentro dos intervalos definidos, mas todas as atualizações subsequentes devem usar a fórmula de correção com base nos vizinhos mais próximos.
- Apenas os três vizinhos mais próximos serão usados para corrigir medições.
- Se não houver vizinhos, a medição fica sempre a mesma.

Novos Sensores:

- Sensores que entram dinamicamente no sistema podem usar medições de outros sensores fora de sua “vizinhança real”, até descobrirem os vizinhos mais próximos.

Limites e Formatação:

- As medições devem ser exibidas com até 4 casas decimais.
- As coordenadas dos sensores no log devem estar no formato (x,y).

Desempenho e Limites:

- Serão testados até **12 sensores simultâneos**. Certifique-se de que o sistema funcione corretamente dentro deste limite.

Erros e Validações:

- Mensagens de erro específicas devem ser exibidas para parâmetros inválidos (tipo ou coordenadas), impedindo o cliente de iniciar.

Sensores sobrepostos:

- Mais de um sensor podem ocupar o mesmo espaço no sistema, de tipos iguais ou diferentes.
- As medições da mesma localização do sensor não são usadas para atualizar as medições, e nem são consideradas vizinhos. Ou seja, se só houver sensores num mesmo local, ninguém atualiza as medições.

Desenvolvimento

O projeto requer a implementação de dois componentes essenciais: um servidor e um cliente, ambos baseados no protocolo TCP. É crucial garantir que ambos os componentes sejam compatíveis **tanto com o IPv4 quanto com o IPv6**, proporcionando flexibilidade na escolha do endereço IP.

Para configurar o servidor corretamente, este deve ser capaz de receber, seguindo rigorosamente essa ordem, o tipo de endereço desejado (v4 para IPv4 ou v6 para IPv6) e um número de porta especificado na linha de comando. Recomenda-se a utilização da porta 51511 para manter a padronização do projeto.

Da mesma forma, os clientes precisam receber, também rigorosamente nessa ordem, o endereço IP do servidor e o número de porta para estabelecer a conexão com sucesso. Certifique-se de que ambos os componentes sejam configurados de acordo com essas diretrizes para uma integração eficaz e a comunicação adequada entre servidor e cliente. A seguir, um exemplo de execução dos programas em dois terminais distintos:

IPv4:

- no terminal 1: `./server v4 51511`
- no terminal 2: `./client 127.0.0.1 51511 -type temperature -coords 1 1`
- no terminal 3: `./client 127.0.0.1 51511 -type air_quality -coords 2 2`

IPv6:

- no terminal 1: `./server v6 51511`
- no terminal 2: `./client ::1 51511 -type humidity -coords 5 6`
- no terminal 3: `./client ::1 51511 -type humidity -coords 6 5`

Sobre Threads:

O uso de threads em servidores é uma abordagem eficiente para lidar com múltiplas conexões simultâneas. Em um servidor multithread, cada cliente conectado é gerenciado por uma thread separada, permitindo que o servidor processe várias requisições de forma paralela. Essa arquitetura tem várias vantagens, com talvez a principal de evitar que a execução de uma operação mais demorada por um cliente bloqueie o atendimento de outros, garantindo maior responsividade do sistema.

No contexto deste trabalho prático, o servidor será responsável por criar uma nova thread para cada cliente que se conecta. Cada thread atuará de forma independente, lidando com as mensagens daquele cliente. Ao mesmo tempo, o servidor principal continuará rodando em sua thread principal, aceitando novas conexões sem interrupções. Essa abordagem permite manter um fluxo contínuo de comunicação entre os sensores e o servidor, assegurando que todas as mensagens sejam tratadas adequadamente, mesmo em cenários com vários sensores conectados simultaneamente. Para aqueles que não conhecem ou desejam relembrar esses conceitos, recomendo essa playlist, em especial os vídeos 5, 6 e 7.

Materiais para Consulta e Dicas:

- Capítulo 2 e 3 do livro sobre programação com sockets disponibilizado no Moodle;
- Playlist de programação com sockets do professor Ítalo Cunha ;
- O guia de programação em rede do Beej (http://beej.us/guide/bgnet/) tem bons exemplos de como organizar um servidor;
- Implemente o trabalho por partes. Por exemplo, implemente o tratamento das múltiplas conexões, depois crie os formatos das mensagens e, por fim, trate os tópicos e as especifidades;
- Procure resolver os desafios da maneira mais simples possível;

Avaliação

O trabalho deve ser realizado individualmente e **deve ser implementado com a linguagem de programação C (não sendo permitida a utilização de C++)** utilizando somente a biblioteca padrão (interface POSIX de sockets de redes). Deve ser possível executar seu programa no sistema operacional **Linux** e **não deve utilizar bibliotecas Windows, como o winsock**. Seu programa deve interoperar com qualquer outro programa implementando o mesmo protocolo **(você pode testar com as implementações dos seus colegas)**. Procure escrever seu código de maneira clara, com comentários pontuais e bem identados. Isto facilita a correção dos monitores e tem impacto positivo na avaliação.

Correção Semi-automática
Seu servidor será corrigido de forma semi-automática por uma bateria de testes. Cada teste verifica uma funcionalidade específica do servidor.

Para a correção os seguintes testes serão realizados **(com IPv4 e IPv6)**:

- Conexão de múltiplos clientes **10%**
- Gerenciamento correto dos vizinhos (descoberta dinâmica) **15%**
- Encaminhamento correto das mensagens para os clientes **15%**
- Atualização correta das medições **15%**
- Impressão correta das mensagens **10%**
- Fechamento de conexão **10%**
- Tratativa dos erros **5%**
- Documentação **20%**

Informações importantes

Cada aluno deve entregar documentação em PDF de até 4 páginas (duas folhas), sem capa, utilizando fonte Arial tamanho 12, e figuras de tamanho

adequado ao tamanho da fonte. A documentação deve discutir desafios, dificuldades e imprevistos do projeto, bem como as soluções adotadas para os problemas.

Atenção: Será dado grande importância a parte da *discussão dos desafios encontrados*, pois as dificuldades ao longo do projeto sempre existem.

A documentação corresponde a 20% dos pontos do trabalho, mas só será considerada para as funcionalidades implementadas corretamente.

Será utilizado um sistema para detecção de código repetido, portanto não é admitido cola de trabalhos.

Será adotada média harmônica entre as notas da documentação e da execução, o que implica que a nota final será 0 se uma das partes não for apresentada.

Cada aluno deve entregar, além da documentação, o código fonte em C e um Makefile para compilação do programa.

Instruções para submissão e compatibilidade com o sistema de correção semi-automática:

- O Makefile deve compilar o cliente em um binário chamado “**client**” e o servidor em um binário chamado “**server**” dentro de uma pasta chamada **"bin"** na raiz do projeto
- Seu programa deve ser compilado ao se executar apenas o comando “make”, ou seja, sem a necessidade de parâmetros adicionais
- A entrega deve ser feita no formato ZIP, com o nome seguindo o seguinte padrão: TP2_MATRICULA.zip

Desconto de Nota por Atraso

Os trabalhos poderão ser entregues até a meia-noite do dia especificado para a entrega. **Não serão aceitos trabalhos com atraso.**