Universidade Federal de Minas Gerais
Departamento de Ciência da Computação

DCC023 Redes de Computadores — TP 1

Sistema supervisório industrial

Alexander Thomas Mol Holmquist

5 de junho de 2022

1 Introdução

Este trabalho apresenta a implementação de um sistema de supervisão de equipamentos de indústria. Este sistema segue a arquitetura cliente-servidor. O servidor é responsável por manter o estado dos equipamentos e seus sensores, e define a semântica de um conjunto de operações com que clientes podem interagir com o estado. O cliente pede ao servidor que execute alguma operação e recebe uma resposta.

Cada equipamento tem um conjunto de sensores, e cada sensor tem um valor. Na vida real, o valor dos sensores representaria alguma grandeza física. Por exemplo, o valor do sensor de temperatura seria a temperatura de um equipamento em um dado momento. Neste trabalho, o valor dos sensores é abstraído, para simplificar. Cada sensor assume um valor aleatório estático entre 0 e 10, determinado quando o sensor é adicionado.

Abaixo apresentamos o conjunto de operações permitidas. O servidor é quem define a semântica de cada uma.

- Adicionar sensores
- Remover sensor
- Listar sensores
- Ler sensores

Cada operação será explicada brevemente na Seção 3.

2 Desafios

A especificação pede para listar desafios e dificuldades, mas não houveram dificuldades consideráveis. Já faz bastante tempo que implemento aplicações parecidas em produção, então foi fácil me adaptar aos requerimentos do projeto. Portanto, a Seção 3 explica a implementação do sistema de uma perspectiva mais direta, sem contrapor soluções com desafios.

3 Implementação

A implementação foi feita em Python 3. O formato das mensagens é especificado de acordo com funções de codificação e decodificação presentes no diretório `monitoring/common/contract`. Tanto cliente quanto servidor utilizam funções desta pasta, especialmente do arquivo `comm.py`. Cada funcionalidade do sistema (adicionar sensores, remover, etc) corresponde a um tipo de requisição definida no arquivo `request.py`. Por exemplo, para adicionar sensores, existe a classe *AddRequest*.

A implementação de todas as funcionalidades por parte do servidor pode ser encontrada no arquivo `monitoring/server/server/server.py`, sob a classe *Server*.

3.1 Adicionar sensores

Formato da mensagem:

```
add sensor <sensor1> <sensor2> ...  <sensorn> in <equipment-id>
```

A função `Server._add_sensors` cuida desta funcionalidade.

3.2 Remover sensor

Formato da mensagem:

```
remove sensor <sensor> in <equipment-id>
```

A função `Server._remove_sensor` cuida desta funcionalidade.

3.3 Listar sensores

Formato da mensagem:

```
list sensors in <equipment-id>
```

A função `Server._list_sensors` cuida desta funcionalidade.

3.4 Ler sensores

Formato da mensagem:

```
read <sensor1> <sensor2> ...  <sensorn> in <equipment-id>
```

A função `Server._read_sensors` cuida desta funcionalidade.

4 Testes

O projeto foi desenvolvido utilizando o paradigma de programação orientada a testes. Para cada funcionalidade, foi primeiramente escrito um teste. Cada teste acorda um servidor em uma thread separada, cria um cliente, e então realiza uma sequência de requisições para exercitar um requerimento específico. O conjunto final de testes de unidade contém 16 funções de testes. Todas as funções de teste podem ser encontradas no arquivo `monitoring/tests/test_client_server.py`.

Os testes fazem uso do framework de testes *Pytest* [1]. Para executá-los, é necessário ter *Pytest* instalado. A partir do diretório `monitoring`, execute o seguinte comando:

```
python3 -m pytest .
```

A Instruções de uso

A interface de linha de comando do programa segue a especificação. Um arquivo de nome `client` pode ser encontrado no diretório principal. Para executá-lo, é necessário ter Python 3 instalado. Veja B para as especificações específicas sob as quais o sistema foi testado.

```
./client 127.0.0.1 51511
   ./client ::1 51511
```

O servidor pode ser executado a partir do arquivo `server`. Também é necessário ter Python 3 instalado.

```
./server v4 51511
./server v6 51511
```

O comando `make` foi incluído para fins de aderência à especificação do trabalho, mas não tem efeito, pois Python é uma linguagem interpretada. `client` e `server` são simplesmente scripts que invocam o interpretador de Python.

O script `test_prof_samples.sh` foi entregado em estado defeituoso. Por favor não tente executá-lo. Ele foi usado para fins de testes manuais.

B Detalhes da máquina local

O sistema foi testado com Python 3.8.10, em uma máquina com a seguinte configuração.

- **Sistema operacional:** Ubuntu 20.04.3 LTS.
- **CPU:** Intel(R) Core(TM) i7-1065G7 @ 1.30GHz-3.90GHz. 1 processador físico, 4 núcleos, 8 threads virtuais.
- **RAM:** 16GB tecnologia DDR4 a 3200MHz.

Referências

[1] Pytest. https://docs.pytest.org/en/7.1.x/. Acessado em 04 de fevereiro de 2022.