Universidade Federal de Minas Gerais
Departamento de Ciência da Computação

DCC023 Redes de Computadores — TP 2

Indústria 5.0

Alexander Thomas Mol Holmquist
16 de julho de 2022

1 Introdução

O objetivo deste trabalho é similar ao do primeiro trabalho da disciplina. Em um ambiente industrial automatizado, é essencial que os dispositivos possam comunicar entre si. Além disso, é essencial que eles possam se comunicar com um servidor central. Arquiteturas em estrela são muito comuns quando dispositivos constrangidos estão envolvidos, pois o servidor central fica encarregado da computação pesada e do armazenamento, além de coordenar a comunicação entre os dispositivos. Com isto em vista, neste trabalho implementamos uma simulação de tal ambiente, com dois componentes:

- Servidor
- Equipamento

O servidor suporta conexão simultânea com até 15 equipamentos, como requerido pela especificação. Nas seções a seguir, discutimos algumas decisões importantes feitas no processo de implementação.

2 Protocolo de mensagens

Nosso protocolo de "aplicação" roda sobre TCP. A especificação determina os campos que cada tipo de mensagem deve conter, mas não determina qual separador utilizar entre as mensagens, e qual codificação (i.e. ASCII ou UTF-8) utilizar. Neste respeito, tomamos as seguintes decisões:

- As mensagens só contém caracteres ASCII – isso facilita a decodificação, pois podemos receber um byte de cada vez no socket.
- As mensagens estão separadas pelo caractere de nova linha
- Campos não presentes em um tipo de mensagem são representados com hífen `-`
- Todo campo, menos o último–*payload* –contém exatamente dois bytes

A implementação da decodificação e codificação das mensagens pode ser encontrada nos arquivos `monitoring/common/comm.py` e `monitoring/common/message.py`.

3 Servidor

Nesta seção, passaremos por alguns dos detalhes que foram necessários levar em conta durante a implementação do Servidor. A principal classe que implementa o servidor se chama `Server`, e pode ser encontrada no arquivo `monitoring/server/server.py`.

3.1 Paralelização

Cada cliente é servido por uma thread diferente. Contudo, as threads não são guardadas em memória. É responsabilidade de cada thread terminar sua conexão com o cliente se necessário, e retornar sem erro. Só foi utilizada uma mutex para exclusão mútua ao acessar a lista de IDs de equipamentos em uso.

3.2 Lista de equipamentos

A lista de equipamentos é mantida através de duas estruturas de dados:

- Um dicionário `equipmentId -> socket`
- Uma lista encadeada de *equipmentId* não utilizados

Os métodos que acessam essas duas estruturas são atômicos, e garantem, através do uso de uma só mutex, `_salt_mutex`, que não existem inconsistências.

3.3 Requisição de informação

O tipo de mensagem ReqInf e ResInf requerem um tratamento especial. Em geral, uma thread só usa o socket de sua própria conexão. Mas no caso específico de ReqInf ou ResInf quando é necessário repassar a mensagem para outro cliente, a thread acessa o método `_send`, que envia a mensagem para o socket associado ao dado ID de equipamento.

3.4 Tratamento de erro

O servidor é resistente a alguns erros mais simples, por exemplo quando um cliente reinicia sua conexão, ou quando um cliente envia uma mensagem inválida: nestes casos, o servidor fecha a conexão com o cliente.

4 Equipamento

Da parte do equipamento, encontramos um desafio muito interessante: como ler entradas da linha de comando ao mesmo tempo que escutamos por mensagens de broadcast? Neste contexto, decidimos que não é necessária a utilização de threads separadas para tratamento de CLI / broadcast. Basta que acrescentemos um timeout na leitura da entrada padrão *stdin*, e do socket em si. Esse controle pode ser observado na função `run` da classe *Client* em `monitoring/client/client.py`.

5 Discussão

6 Conclusão

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. Só são necessárias as bibliotecas padrões de Python 3. Veja B para as especificações específicas sob as quais o sistema foi testado.

```
./client localhost 51511
```

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

```
./server 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.

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.