PROCESSO DE SELEÇÃO WAY2 TECNOLOGIA

TESTE 01 – DESENVOLVEDOR C#

**O PROBLEMA**

Escreva um programa do tipo console application em que possamos nos conectar ao socket TCP no servidor especificado, ler alguns dados de medição a partir do protocolo de comunicação definido abaixo e salvar estes em um arquivo CSV.

**Obs**: No socket, existe um medidor fake que irá responder aos pedidos do seu programa através das portas disponibilizadas.

**Endereço do servidor:** 40.121.85.141

**Portas disponíveis:** será disponibilizada junto com o teste para cada candidato.

**REQUISITOS**

O programa deverá aceitar o seguinte argumento em linha de comando:

* listaDePedidos: O nome de um arquivo contendo os pedidos a serem executados, onde cada pedido é uma linha do arquivo com os seguintes campos separados por espaço simples.
  + **ip**: String com o endereço IPv4 (xxx.xxx.xxx.xxx)
  + **porta**: Int com o range válido de uma porta TCP
  + **indiceInicial**: Int com índice do primeiro registro a ser lido
  + **indiceFinal**: Int com o índice do último registro a ser lido

**Obs**: Fique a vontade para decidir se os pedidos serão feitos em serial ou em paralelo e se a comunicação será assíncrona ou síncrona.

**Exemplo de comando:**

|> [NomeDoExecutavel].exe listaDePedidos.txt

O protocolo de comunicação consiste em uma sequência de troca de frames de bytes. Dado que o aplicativo é o client, ele deve enviar pedidos de coleta ao medidor (Server), que por sua vez, irá responder aos pedidos.

Cada frame tem um código de função. Abaixo as funções válidas:

* 0x01 – Ler Número de série
* 0x02 – Ler o status dos registros: índice do registro mais antigo e o índice do registro mais novo
* 0x03 – Definir o índice do registro para ser lido
* 0x04 – Ler data e hora do registro atual
* 0x05 – Ler valor de energia do registro atual

Os frames dos bytes:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Frame Header | Comprimento da mensagem | Código da Função | Dados | Checksum |

Frame Header Byte – 1 Byte com valor fixo de 0x7D

Comprimento da mensagem – 1 Byte com valor igual ao número de bytes do campo Dados.

Código da Função – 1 Byte com valor igual ao código da função. Nos 7 bits menos significativos estão o código do pedido que o frame faz referência (0x01-0x05 definido acima). O bit mais significativo tem valor 0 se o frame é um pedido (do cliente), e tem valor 1 se o frame é uma resposta (do medidor). O código de função de 0xFF significa um erro, podendo ser enviado pelo cliente ou pelo medidor.

Bytes de Dados – pode ter 0x00 até 0xFF Bytes, dependendo do código de função.

Checksum – 1 Byte, calculado fazendo-se o OU EXCLUSIVO (XOR) de todos os bytes do frame, com exceções do Frame Header e do Checksum byte.

**Funções**

**Ler Número de Série:**

Pedido: código de função 0x01, zero bytes de dados

Exemplo: 7D 00 01 01

Resposta: código de função 0x81, null terminated ASCII string nos bytes de dados

Exemplo: 7D 08 81 41 42 43 44 45 46 47 00 C9 - contém string ‘ABCDEFG’

**Ler Registro Status:**

Pedido: código de função 0x02, zero bytes de dados

Exemplo: 7D 00 02 02

Resposta: código de função 0x82, dois unsigned 16-bit integers nos bytes de dados

Exemplo: 7D 04 82 01 2C 02 58 F1 – contém o índice mais antigo (com valor 300) e o índice mais novo (com valor 600)

**Definir índice para ser Lido:**

Pedido: código de função 0x03, um unsigned 16-bit integer nos bytes de dados

Exemplo: 7D 02 03 01 7C 7C – contém índice 380

Resposta: código de função 0x83 e um byte que indica se a definição foi um sucesso. 0x00 indica sucesso, outro valor indica falha.

Exemplo: 7D 01 83 00 82 – contém sucesso

**Ler Data e Hora do Registro Atual:**

Pedido: código de função 0x04, zero bytes de dados

Exemplo: 7D 00 04 04

Resposta: código de função 0x84, com 5 bytes de dados: primeiros 12 bits significam o ano, próximos 4 bits significam o mês, próximos 5 bits significam o dia, próximos 5 bits significam a hora, próximos 6 bits significam o minuto, próximos 6 bits significam os segundos, e os últimos 2 bits não são usados (ignorar a valor)

Exemplo: 7D 05 84 7D E1 BC 59 2B D3 – contém data e hora 2014-01-23 17:25:10

**Ler Valor de Energia do Registro Atual:**

Pedido: código de função 0x05, zero bytes de dados

Exemplo: 7D 00 05 05

Resposta: código de função 0x85, com 4 bytes de dados (ponto flutuante de precisão simples, conforme IEEE 754)

Exemplo: 7D 04 85 41 20 00 00 E0 – contém o valor 10,0

**Erro:**

Pedido e Resposta: código de função 0xFF, zero bytes de dados

Exemplo: 7D 00 FF FF

Se um lado receber um frame com o checksum errado, ele deve enviar um frame de Erro. Se um lado receber um frame de Erro, ele deve reenviar o último frame enviado. Se um lado receber um frame com dados com tamanho inesperado ou com formato inválido, ele deve enviar um frame de erro. Se o medidor receber um código de uma resposta, ele enviará um frame de erro.

A data e hora e o valor de energia (funções 0x04 e 0x05) só terão significado se o índice para ser lido foi definido com sucesso através do comando 0x03. Se o medidor retornar uma resposta com falha, o registro é considerado faltante. Neste caso, data e hora e valor não devem ser lidos. Normalmente isso acontece ao tentar ler índices fora do range dos registros informados pela função 0x02. No entanto, é possível acontecer com alguns registros dentro do range informado. O medidor nunca vai retornar sucesso como resultado da função 0x03 quando um índice estiver fora do range.

**A leitura deve seguir a seguinte sequência:**

1) conectar ao socket

2) ler o número de série

3) ler o registro status

4) determinar quais registros serão lidos de acordo com indiceInicial, indiceFinal e o status dos registros.

5) tentar ler a data, hora e valor de cada registro

6) desconectar do socket

7) escrever dados no arquivo

Considerando que cada registro contém: um índice, uma data e hora e um valor de energia (em kWh), o aplicativo deverá ler o número de série do medidor e a sequência de registros coletados, caso eles existam no medidor.

Após a leitura, o programa deve escrever os dados coletados num arquivo CSV com ‘;’ usado como separador. A primeira linha deve conter o número de série, e o resto dividido em 3 colunas.

1. Índice do registro
2. Data e hora do registro, no formato ‘YYYY-MM-DD HH:MM:SS’,
3. Valor do registro, usando ‘,’ como separador de casas decimais e com duas casas decimais (ex. 1234,56). Os valores devem ser arredondados para o decimal mais próximo com metade arredondado ao valor par (ex. 1,014 -> 1,01; 1,015 -> 1,02; 1,016 -> 1,02; 1,025 -> 1,02).

O servidor só aceita uma conexão por vez. Não há um comando de desconexão, logo, é necessário esperar o servidor fazer reset antes de conectar de novo. O servidor possui um timeout de 60 segundos, onde ele fará o reset e estará pronto para receber uma nova conexão.

Considere os problemas possíveis da rede para que o programa fique o mais estável possível: perda de pacotes, resposta lenta, bytes incorretos, perda de conexão, etc.

**PREMISSAS DO PROJETO**

* Ser escrito em c# 4.5 ou superior, console application
* Solução no VS2015 ou superior
* Deve estar pronto para ser compilado **no Visual Studio** e nada mais.

**VAMOS AVALIAR**

* Clareza e qualidade do código
* Cobertura e qualidade dos testes
  + Escolha da estrutura de dados
  + Cuidados com a memória e quantidade de I/O
* Caso seja optado pela opção em paralelo, avaliaremos à estrutura utilizada para tratar a concorrência.
* Caso seja optado pela opção com comunicação assíncrona, avaliaremos a utilização do pattern async/await do .NET framework.