UNIVERSIDADE FEDERAL DE MINAS GERAIS

Sistemas distribuídos para automação

Trabalho prático

Professor Luiz Luiz Themystokliz Sanctos Mendes

Estevão Coelho Kiel de Oliveira - 2016119416
Lucas Costa Souza - 2018013763

9 de agosto de 2021 - 2021/1

Conteúdo

1 Introdução

Este relatório refere-se a documentação do projeto desenvolvido na disciplina de *Sistemas distribuídos para automação*, o mesmo diz respeito a uma aplicação multithread que tem como objetivo a integração de um servidor de sockets e um cliente OPC. O servidor de sockets se comunica com um cliente de sockets instalado em outra maquina em uma mesma rede local e o cliente OPC se comunica com um servidor OPC.

O cliente de sockets funciona como um MES (Manufacturing Execution System), que basicamente tem o objetivo de capturar de maneira contínua informações da planta industrial e da área de gestão de negócios da empresa para a geração de scheduling. Já o servidor OPC, do tipo clássico, se comunica diretamente com os CLPs (Controladores Lógicos Programáveis) que são os responsáveis pelo controle das linhas de envase do processo de produção de bebidas não alcoólicas.

Desse modo, o MES deve ser abastecido de maneira contínua com os dados do status da planta que estão disponíveis no servidor OPC e os CLPs devem receber valores de setup de produção provenientes do MES, esse valores correspondem a cada tipo diferente de produto a ser envasado.

2 Desenvolvimento

2.1 Visão global

A Figura 1 a seguir é uma representação visual da aplicação desenvolvida inserida no seu contexto de funcionamento.

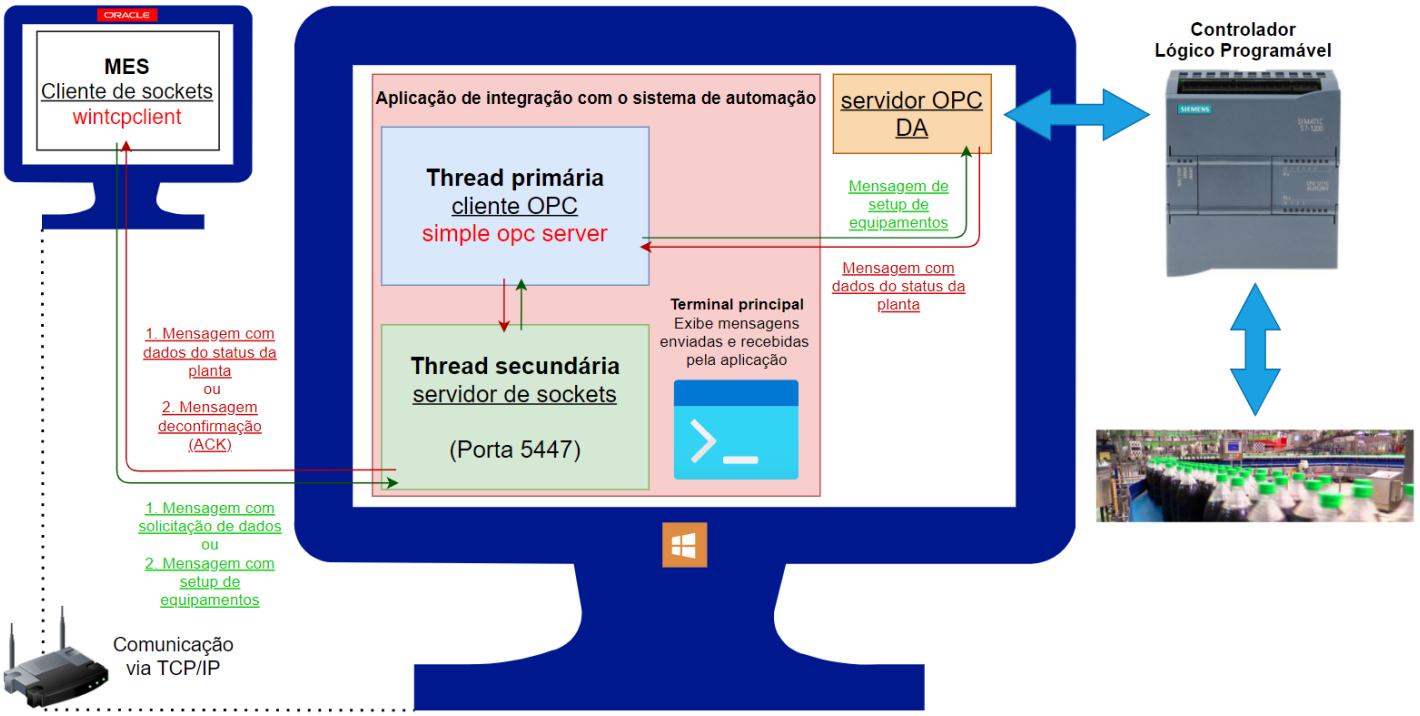

Figura 1: Representação visual da aplicação inserida em seu contexto de funcionamento

O contexto em que estamos inseridos é o de uma indústria de bebidas não alcoólicas e a mesma se encontra no estágio de implementação de uma solução do tipo MES (Manufacturing Execution System). O MES tem a função de gerar o scheduling de produção e o setup dos equipamentos das linhas de envase. Sua simulação é feita a partir de um cliente de sockets fornecido pelo professor que se encontra em uma pasta com o nome wintcpclient. Na prática MES estaria instalado em um Servidor Dell PowerEdge T340 com o S.O. Solaris da Oracle. Podemos ver na ilustração que o mesmo está conectado a outro computador, que roda a nossa aplicação e o servidor OPC DA, em uma mesma rede local por conexão sem fio e comunicação via TCP/IP. O MES primeiro se conecta com a aplicação por meio da thread secundária servidor de sockets via porta 5447 e então começa a disparar e receber mensagens. Caso a conexão entre os dois seja interrompida por qualquer motivo a aplicação de software descarta qualquer mensagem pendente de envio ou recebimento e tenta reconectar-se automaticamente com o servidor de sockets.

As mensagens que o MES envia para a aplicação são de dois tipos. O primeiro é uma mensagem que solicita os dados do status da planta recolhidos em um servidor OPC DA por uma thread primária da aplicação cliente OPC. Podemo ver seu formato na Tabela 1. Já o segundo tipo diz respeito a uma mensagem com o setup de equipamentos, a aplicação por meio da thread cliente OPC irá escreve-los no servidor OPC DA que por sua vez irá envia-los aos CLPs (Controlador Lógico Programável). Esses valores correspondem a cada tipo diferente de produto a ser envasado. Podemo ver seu formato na Tabela 2

Campo	Tipo	Tamanho	Formato	Observação
Código da mensagem	Inteiro	2	00	Sempre 00.
Número sequencial da mensagem	Inteiro	7	NNNNNNN	Vai de 1 até 9999999.

Exemplo de mensagem enviada	00/0005577

Tabela 1: Mensagem de solicitação de dados enviada pelo MES para a Aplicação de software

Campo	Tipo	Tamanho	Formato	Observação
Código da mensagem	Inteiro	2	77	Sempre 77.
Número sequencial da mensagem	Inteiro	7	NNNNNNN	Vai de 1 até 9999999.
Número da linha de envase	Inteiro	2	NN	Bucket Brigade.Int1
Set-point de velocidade da esteira (cm/s)	Real	6	NNNN.N	Bucket Brigate.Real4
Set-point da máquina de enchimento (m3/min)	Real	6	NNNN.N	Bucket Brigade.Real8
Set-point da máquina separadora (cm/s)	Inteiro	5	NNNNN	Bucket Brigade.Int4

Exemplo de mensagem enviada	77/0007718/02/0040.0/0030.0/00020

Tabela 2: Mensagem com setup de equipamentos enviada pelo MES para a Aplicação de software

Para toda mensagem enviada pelo MES existe uma resposta correspondente vinda da Aplicação de integração com o sistema de automação da fábrica. Para a mensagem que solicita os dados do status da planta a aplicação, por meio do servidor sockets, retorna imediatamente uma mensagem com dados do status da planta vinda diretamente do servidor OPC. Já para mensagem com o setup de equipamentos a aplicação retorna uma mensagem de confirmação (ACK). Podemos ver o formato das duas respectivamente na Tabela 3 e 4.

Campo	Tipo	Tamanho	Formato	Observação
Código da mensagem	Inteiro	2	99	Sempre 99.
Número sequencial da mensagem	Inteiro	7	NNNNNNN	Vai de 1 até 9999999.
Produção acumulada na linha de envase 1 (l)	Real	7	NNNNN.N	Random.Real4
Produção acumulada na linha de envase 2 (l)	Real	7	NNNNN.N	Saw-toothed Waves.Real4
Nível do tanque de produto 1 (cm)	Inteiro	3	NNN	Random.Int1
Nível do tanque de produto 2 (cm)	Inteiro	3	NNN	Randon.Int2
Downtime linha de envase 1 (min) no turno	Inteiro	3	NNN	Randon.Int4
Downtime linha de envase 2 (min) no turno	Inteiro	3	NNN	Saw-toothed Waves.Int2

Exemplo de mensagem enviada	99/0023718/02043.5/00941.3/077/420/023/007

Tabela 3: Mensagem com dados do status da planta enviada pela Aplicação de software para o MES

Campo	Tipo	Tamanho	Formato	Observação
Código da mensagem	Inteiro	2	22	Sempre 22.
Número sequencial da mensagem	Inteiro	7	NNNNNNN	Vai de 1 até 9999999.

Exemplo de mensagem enviada	00/0005577

Tabela 4: Mensagem de confirmação (ACK) enviada pela Aplicação de software para o MES

Vale ressaltar que todas as mensagens produzidas são estruturadas como cadeias de caracteres ASCII onde os campos são delimitados por "/"e que todas as mensagens são sequenciais entre si, ou seja, elas não se incrementam isoladamente, a nova depende do número sequencial da anterior mesmo que esta seja de outro tipo. Os números sempre são incrementados pelo emissor das mensagens e quando a contagem máxima for atingida o contador é zerado. Podemos observar um exemplo de troca de mensagens na Tabela 5 a seguir.

Mensagem	Origem	Número sequencial
Requisição de status da planta	MES	0000001
Mensagem com dados de processo	Aplicação de software	0000002
Requisição de status da planta	MES	0000003
Mensagem com dados do processo	Aplicação de software	0000004
Envio de setup de produção	MES	0000005
ACK	Aplicação de software	0000006
...	...	0000007

Tabela 5: Exemplo de mensagens trocadas entre o MES e a Aplicação de software

Por fim, todas as mensagens que trafegam entre a aplicação de software e o MES são exibidas no terminal principal.

Já no que se diz respeito ao cliente OPC inserido na aplicação de software, que foi baseado na versão modificada do Simple OPC Client apresentado pelo professor e que está funcionando como a thread primária do programa desenvolvido, o mesmo executa as seguintes ações.

Ler do servidor OPC, que está rodando no mesmo computador Windows que a aplicação, de maneira assíncrona via notificações de callback as variáveis que correspondem ao status das linhas de envase da fábrica. Esse valores lidos sempre devem estar atualizados para que quando o MES pedir um dado ele seja o mais atual possível. A taxa de leitura é de 1000ms.

Escrever no servidor OPC, de maneira síncrona os valores que dizem respeito aos dados de setup de produção recebidos do MES.

Por fim, a aplicação de software também deve exibir todos os valores lidos e escritos pelo cliente OPC do servidor OPC no Terminal principal do programa.

A nossa aplicação multithread foi desenvolvido no Visual Studio Community da Microsoft na linguagem C++. A seguir temos uma explicação mais detalhada do funcionamento de cada parte da aplicação.

As threads do nosso processo compartilham memória por meio de variáveis globais, toda e qualquer alteração nelas é feita com a proteção de objetos do kernel do tipo mutex para evitar conflitos de leitura e escrita na seção crítica.

Sempre que a tecla 's' é precionada no MES o mesmo solicita a escrita dos valores de setup de equipamentos no Servidor OPC. Caso uma mensagem desse tipo chegue até o servidor de sockets o mesmo guarda os valores em uma variável global e sinaliza um objeto do tipo evento. Esse objeto ativa um trecho de código responsável por escrever no Servidor OPC DA os dados de setup dos equipamentos.

O funcionamento do módulo cliente OPC e do cliente de sockets é independente entre si pela maneira como a aplicação foi estruturada em programação multithread.

2.2 Thread servidor de sockets

A thread servidora de sockets é criada pela thread primária responsável pelo Cliente OPC da aplicação. O servidor de sockets é responsável pela comunicação entre o cliente de sockets (fornecido pelo professor no arquivo wintcpcliente). O cliente de sockets simula o funcionamento do MES que é responsável por mandar mensagens periódicas solicitando dados da planta e por mandar mensagens não periódicas (enviadas quando a tecla s é precionada no teclado do cliente de sockets) com os setups dos equipamentos.

Primeiro a thread carrega a biblioteca winsock versão 2.2 por meio da função WSAStartup(), depois ela define o endereço do servidor e a porta para a conexão (no caso a 5447), em seguida cria o socket para a conexão com o servidor por meio da função socket(), logo após a função bind() é responsável por vincular o socket a porta definida e por fim a função listen() coloca a porta TCP em modo de escuta.

Depois desse processo entramos em um loop que começa com a função accept(), essa função aguarda a conexão de um cliente de sockets e então continua a execução da aplicação. Então o programa entra em outro loop mais interno e fica trocando mensagens com o cliente por meio das funções recv() e send().

A função recv() espera uma mensagem do MES, dependendo do seu tipo da mensagem o servidor de sockets é responsável por retornar uma outra mensagem correspondente como visto anteriormente. Caso o loop mais interno de recv() e send() termine a thread volta para o loop mais externo e aguarda outra conexão por meio da função accept().

2.3 Thread Cliente OPC

A thread principal da nossa aplicação é baseada na versão "Simple OPC Client"desenvolvida por Philippe Gras (CERN) e pode ser encontrada na data atual em http://pgras.home.cern.ch/pgras/OPCClientTutorial/. Esse cliente utilizado como base ainda foi alterada pelo professor Luiz T. S. Mendes do DELT/UFMG em 15 Setembro de 2011, onde ele introduziu duas classes em C++ para permitir que o cliente OPC requisite callback notifications de um servidor OPC. Também foi necessário incluir um loop de consumo de mensagens no programa principal para permitir que o cliente OPC processe essas notificações. As classes em C++ implementam as interfaces para o cliente do OPC DA 1.0 IAdviseSink (removida na nossa aplicação) e do OPC DA 2.0 IOPCDataCallback. Algumas funções para inicialização e cancelamento das funções também foram implementadas e alteradas pela dupla.

2.3.1 Leitura do servidor

A ação de leitura de dados do servidor MatrikonOPC Server for Simulation é realizada pela estrutura while(TRUE) contido na função main. As funções de GetMessage(), TranslateMessage() e DispatchMessage() foram mantidas do programa "Simple OPC Client"com alterações do Professor. Elas são responsáveis por captar uma mensagem do Windows (pelo GetMessage) quando um método de uma thread secundária é chamado, e é responsável também por consumir esta mensagem (pelo DispatchMessage).

Para a leitura dos dados do servidor foram implementados 2 novos métodos da classe SOCDataCallback (sendValues() e sendHandles()), que é responsável por cuidar da chamada de DataCallback do sistema quanto à atualização de dados dentro do Servidor OPC; bem como a adição de 2 novos atributos à esta classe (dadosLeitura e handlesLeitura), responsáveis por armazenar os respectivos valores e handles das variáveis lidas pelo Cliente OPC do Matrikon.

A função sendValues() é responsável por enviar os dados lidos, armazenados no atributo dadosLeitura do SOCDataCallback, para a função main presente na thread principal. A função sendHandles, também parecida com a anterior, é responsável por enviar os handles dos dados lidos, armazenados no atributo handlesLeitura do SOCDataCallback, para a função main presente na thread principal.

Então, dentro da estrutura do while são chamadas estas 2 funções, onde a thread primária recebe os dados lidos pela classe SOCDataCallback.

Dentro do programa principal, também foi adicionada uma nova função (decode()). Esta função é responsável por descodificar os dados recebidos do Matrikon, transformado-os do formado VARIANT para uma mensagem char no formato especificado pelo Servidor de Sockets para o envio dos dados de *status* da planta.

A função faz a leitura os dados lidos em formato VARIANT, e utilizando a função VarToStr() que já estava contida no programa original alterado pelo professor "Simple OPC Client", transforma os dados em uma string. Em seguida, utilizando a função sprintf(), o programa faz o rearranjo dos dados com a inclusão das barras entre os valores e especificando as casas decimais e os números exibidos da maneira especificada na mensagem que o Servidor de Sockets deve enviar.

2.3.2 Escrita no servidor

A ação de escrita de dados no servidor Matrikon também é realizada pela estrutura while(TRUE) presente na função main() do programa principal. Porém, esta parte da função é ativada por um evento inserido dentro da thread de Servidor de Sockets.

Quando o Servidor de Sockets recebe uma mensagem do MES contendo o *setup* de equipamentos, ele ativa um evento que é esperado dentro do while contido na função main da thread Cliente OPC. Neste sentido, o programa detecta que este evento foi acionado, ativando uma variável que coloca o programa dentro de uma estrutura if responsável por formatar os dados no formato VARIANT e enviando-os para o Servidor OPC.

Dentro da estrutura if, o programa faz a leitura dos dados recebidos do MES de *setup* de equipamentos, transforma os dados do formato de string para valores em formatos float. Em seguida, o programa utiliza-se de uma variável auxiliar (aux) do tipo VARIANT, e traduz cada dado do formato float para a citada anteriormente, com seu respectivo tipo de valor (Int1, Int4, Real4 e Real8). Então, ela envia os valores traduzidos para o Servidor OPC, pela função WriteItem implementada no programa principal.

A função WriteItems foi implementada a partir da função ReadItem que já existia no programa base "Simple OPC Client". Ela cria um ponteiro para a interface IOPCSyncIO presente no componente OPC Group Object, referenciando os métodos de leitura e escrita síncrona do OPC. Em seguida, ele chama a função Write daquela interface, que é responsável por ler síncronamente os dados contidos no grupo que foi instanciado inicialmente (o ponteiro para esta interface é passado para a função WriteItem como um atributo). A função explicita em situações de erro de escrita. Do contrário, a função somente libera as memórias alocadas dentro dela e libera a referência à interface IOPCSyncIO.

3 Como executar o programa

O arquivo trabalho_sda_estevao_e_lucas.zip deve ser descompactado, e dentro da pasta a solução SimpleOPCClient.sln deve ser aberta.

Para que o programa compile e funcione corretamente é necessário incluir a biblioteca Winsock2 (Ws2_32.lib) no projeto. Para isso deve-se ir no Visual Studio Community Edition em Project → Properties → Configuration Properties → Linker → Input e no campo Additional Dependencies adicione a biblioteca Ws2_32.lib.

Após isso deve-se compilar e rodar o arquivo. Vale ressaltar que as vezes ele não funciona de primeira, por isso é importante encera-lo e executa-lo novamente. Quando o mesmo estiver rodando pode-se disparar o MES, a conexão será realizada e todo o ambiente estará preparado.

4 Resultados

O Terminal Principal da aplicação e o MatrikonOPC Server podem ser vistos em execução na Figura 2. É possível verificar o correto funcionamento de todas as funcionalidades da Aplicação de software e a confirmação da escrita dos valores no Servidor OPC DA por meio do programa MatrikonOPC Server.

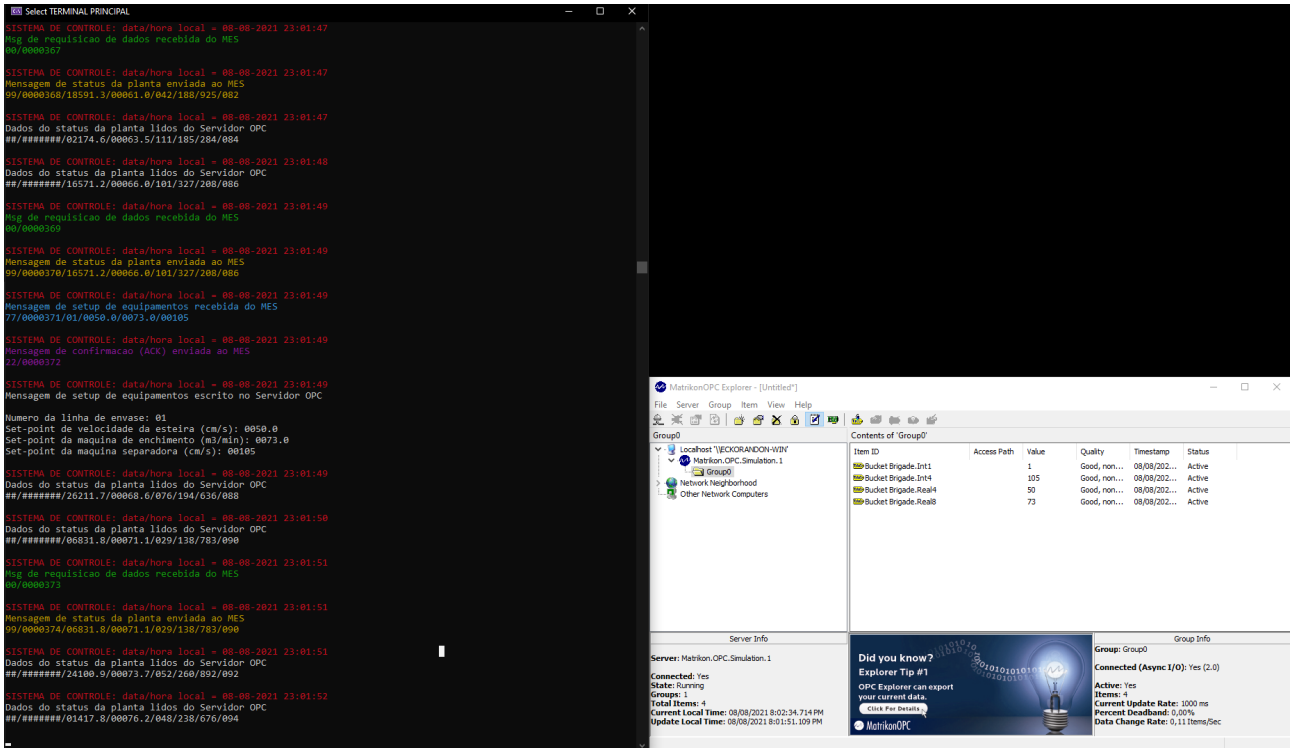

Figura 2: Resultados obtidos

5 Conclusão

Todas as definições do projeto foram atendidas e estão funcionando perfeitamente. Todos os valores transmitidos e recebidos estão sendo exibidos no Terminal Principal de maneira detalhada e organizada.