

Rede Física e Lógica Cliente / Servidor

FEELT / UFU
Uberlândia

Luiz Carlos ROMA JÚNIOR *(98101)*

Disciplina Redes Locais Industriais

RLI
9º período

Ano Escolar 2013 - 2014



Agradeço a orientação do Sr. Alexandre Rodrigues de Sousa durante todo o processo de desenvolvimento, sendo disposto e prático em horários de aula e outros horários.

Agradeço igualmente o Prof. Dr. José Jean-Paul Zanlucchi de Souza Tavares pela proposta de módulo, cujo qual tem direta aplicação em meios industriais e pôde me possibilitar o aprendizado de várias ferramentas.

Introdução

Um Controlador Lógico Programável ou Controlador Programável, conhecido também por suas siglas CLP ou CP e pela sigla de expressão inglesa PLC (*Programmable Logic Controller*), é um computador especializado, baseado em um microprocessador que desempenha funções de controle através de softwares desenvolvidos pelo usuário (cada CLP tem seu próprio software). Geralmente as famílias de Controladores Lógicos Programáveis são definidas pela capacidade de processamento de um determinado número de pontos de Entradas e/ou Saídas (E/S).

Controlador Lógico Programável segundo a ABNT (*Associação Brasileira de Normas Técnicas*), é um equipamento eletrônico digital com hardware e software compatíveis com aplicações industriais. Segundo a NEMA (*National Electrical Manufacturers Association*), é um aparelho eletrônico digital que utiliza uma memória programável para armazenar internamente instruções e para implementar funções específicas, tais como lógica, sequenciamento, temporização, contagem e aritmética, controlando, por meio de módulos de entradas e saídas, vários tipos de máquinas ou processos.

Um CLP é o controlador indicado para lidar com sistemas caracterizados por eventos discretos (*SEDs*), ou seja, com processos em que as variáveis assumem valores zero ou um (ou variáveis ditas digitais, ou seja, que só assumem valores dentro de um conjunto finito). Podem ainda lidar com variáveis analógicas definidas por intervalos de valores de corrente ou tensão elétrica. As entradas e/ou saídas digitais são os elementos discretos, as entradas e/ou saídas analógicas são os elementos variáveis entre valores conhecidos de tensão ou corrente.

Com esses conceitos em mente, esse trabalho vem para fazer um apanhado geral da metodologia de conexão desse tipo de hardware com um programa de computador - nesse caso a linguagem usada será C++ em compatibilidade com a *.dll* do CLP, leitura e escritura de dados e, ao final, como fazer a consulta de alguns dados lidos desse CLP através de um servidor WEB - desenvolvimento em PHP, Apache e PostgreSQL.

1- Sobre o CLP e .dll

Primeiramente, é importante destacar as configurações básicas do CLP utilizado, assim como os parâmetros de comunicação, arquivos disponíveis, princípios, etc.

O hardware utilizado em laboratório é um CLP considerado didático - não tão robusto e tampouco de alto custo como os industriais - mas que representa bastante esse tipo de dispositivo utilizado em campo. Ele é um modelo ZAP900 fabricado pela HI Tecnologia.



Figura 1. : Imagem do CLP usado em laboratório.

Para a ligação desse CLP com o computador é preciso de um arquivo de pilotagem e de especificação afim de estabelecer a conexão e as configurações dessa conexão. Nesse projeto foi usada a .dll do CLP fornecida pela empresa e um arquivo .cfg também fornecido, os quais continham :

- *Porta de comunicação ;*
- *Baud rate ;*
- *Paridade ;*
- *Stop bits ;*
- *Data bits ;*
- *RS485 control ;*
- *Suporte para comunicação via data radio ;*
- *Etc.*

Antes de citar as configurações usadas para estabelecer a conexão do CLP com o notebook utilizado, é importante dizer que essa conexão foi feita através de um cabo adaptado USB / Serial, cuja foto é mostrada logo abaixo :



Figura 2. : Cabo USB / Serial utilizado para fazer a conexão entre o CLP do laboratório e o notebook.

Esse cabo, cujo custo aproximado gira em torno de R\$ 40.00, vêm acompanhado de um cd de instalação de *drivers* para diversos tipos de sistemas operacionais. A versão do sistema utilizado é Windows 7 64 bits e, portanto, o *driver* correspondente à essa versão foi instalado.

Após essa instalação do *driver* no Windows, é criada uma porta serial COM virtual - a porta é simulada - e é possível, então, mudarmos algumas definições dessa porta. Para esse exemplo, a porta usada foi a COM 8 e as configurações dessa porta - as quais também foram modificadas no arquivo de configuração *ScpHi.cfg* são :

- *pcpsPort* = 8 ;
- *pcpsBaud* = 9600 ;
- *pcpsParity* = 0 ;
- *pcpsStopBits* = 1 ;

Apenas os dados citados foram aqueles modificados no arquivo de configuração. Após salvar esse arquivo, a próxima parte foi o estudo das funções de comunicação do CLP através da .dll fornecida pela empresa HI Tecnologia, assim como entender um pouco mais sobre os tipos de variáveis manipuladas pelo CLP utilizado.

2- Documentação do CLP

2-1. Funções básicas

Com toda a documentação do CLP em mãos - fornecida junto com o material físico - foi possível identificar algumas funções-chaves para o desenvolvimento do software de comunicação. Essa documentação explica todas as funções disponíveis dentro da .dll de comunicação com o CLP. Sabendo a estrutura dessas funções, pode-se desenvolver, utilizando essa .dll e as bibliotecas corretas, uma linguagem de programação conveniente afim de manipular dados com o dispositivo.

DLL (Dynamic-Link Library), que do inglês, significa Biblioteca de Vínculo Dinâmico. Os formatos de arquivos para DLL são os mesmos dos arquivos executáveis para Windows. Assim como os executáveis (EXE), as DLL podem conter códigos, dados, e recursos (ícones, fontes, cursores, entre outros) em qualquer combinação.

Dentre as funções encontradas no manual, pode-se destacar :

- *SCPConfigComChannel* (Abre uma tela para configuração de todos os parâmetros de comunicação utilizados pelo driver. Esta tela é contextual e dependendo do tipo de recurso de comunicação selecionado (ex: porta serial ou Ethernet), abas adicionais são apresentadas para configurações específicas da opção selecionada) ;
- *SCPCheckConnection* (Envia um frame de teste de conexão para o controlador conectado a porta serial aberta pelo driver. Esta função permite detectar a presença de controlador e avaliar se o link de comunicação está operacional) ;
- *SCPOpenPort* (Abre o canal de comunicação serial, alocando e configurando a porta serial especificada. Obtém do arquivo CfgFileName os parâmetros de configuração necessários para programar a porta serial do computador. Após a execução desta função com sucesso o driver está apto para trocar frames com o controlador conectado. Terminado o processo de comunicação, utilize a função ScpClosePort para liberar o canal de comunicação para o Windows) ;
- *SCPClosePort* (Fecha a porta de comunicação utilizada pelo driver, disponibilizando este recurso novamente para o Windows) ;
- *SCPReadData* (Solicita ao controlador os valores das variáveis especificadas na função. O usuário deve especificar o tipo de variável (R, M ou D), identificar o número da variável inicial e a quantidade de variáveis a serem obtidas a partir desta. Os valores das variáveis solicitadas são transferidos para o buffer de dados fornecido. É de responsabilidade do usuário prover o buffer de recepção com o espaço necessário para receber todos os dados solicitados) ;
- *SCPWriteData* (Transfere para o controlador o conteúdo do buffer de variáveis especificadas na função. O usuário deve especificar o tipo de variável (R, M ou D), identificar o número da variável inicial e a quantidade de variáveis a serem transferidas a partir desta) ;

Antes de passar ao próximo tópico que indica algumas características dos tipos de variáveis lidas/escritas pelo CLP, é importante notar que essas funções obedecem um tipo de função protótipo que, no caso de C++, é dada por :

tipo_retorno_função WINAPI SCP**Função** (*tipo_var1* Var1, *tipo_var2* Var2, etc ...)

2-2. Tipos de variáveis manipuladas

Esse CLP trabalha com 3 tipos de variáveis : R, M e D. As variáveis tipo R são variáveis de estado, ou chamadas de variáveis de Contato Auxiliar. As variáveis M são variáveis de Memórias Inteiras e as variáveis tipo D são variáveis de Memórias reais.

A seguir duas tabelas retiradas da documentação da .dll do CLP :

<u>Tipo da Variável</u>	<u>Tipo associado na linguagem C</u>	<u>Tamanho</u>
R	unsigned char	1 byte (8 bits)
M	short int	2 bytes (16 bits)
D	float	4 bytes (32 bits)

Figura 3. : Tabela mostrando os tipos de variáveis à serem declaradas em C++.

<u>TypeVar</u>	<u>Tipo de variável</u>	<u>ID</u>	<u>Tamanho</u>	<u>Faixa de valores</u>	
				Min	Max
0	Contato auxiliar	R	1 byte	OFF(0)	ON(255)
1	Memórias inteiras	M	2 bytes	-32768	+32767
2	Memórias reais	D	4 bytes	-10E-38	+10E+38

Figura 4. : Limite de valores para as variáveis aceitas.

2-3. Estrutura do programa

Com esses conceitos de base, o software foi pensado da seguinte maneira :

- Carregamento automático da .dll do CLP ;
- Monitoramento de estados variáveis tipo R em forma de LED's ;
- Monitoramento de estados variáveis tipo M em barras de rolagem (track bars) ;
- Monitoramento de estados variáveis tipo D em barras de rolagem (track bars) ;
- Clock para o set do tempo de leitura cíclica ;
- Botões para checagem de conexão e começo de leitura ;
- Textbox afim de fazer aquisição de dados escritos pelo usuário e gravação das informações no CLP ;
- Botão para repetição de valores de leitura nos valores de escritura ;
- Opções para escritura em um banco de dados ;

Em termos práticos, o programa carregaria a *.dll* (biblioteca dinâmica) no início do programa e teria um tempo préconfigurado de clock para a *leitura cíclica*. O usuário clica em << *Checar Conexão* >> afim de saber se a conexão está correta com o CLP e inicia a leitura dos dados - monitoramento das variáveis em tempo real pela interface gráfica - através de um botão chamado << *Iniciar Leitura* >>. Se ele quiser escrever alguns dos dados e alterar estados das variáveis, ele clica em um botão que guarda os valores das variáveis em textbox's - << *Manter Valores* >> - e, logo em seguida, em << *Escrever Dados* >>.

O escrita dos valores lidos no banco de dados (criado em SQL) só se dá após o clique do usuário no botão << *Conectar Banco de Dados* >>. Essa escrita pode ser de forma << *Completa* >> ou << *Simplificada* >> - ou o programa escreve todas as variáveis toda vez que o clock pulsar ou o programa escreve apenas as variáveis modificadas no CLP.

Afim de esclarecer um pouco mais a estrutura do programa, vejamos sua interface gráfica :

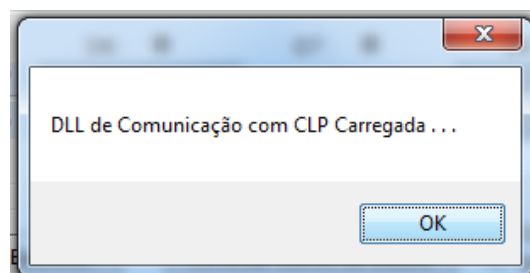


Figura 5. : Janela inicial mostrando que a DLL foi carregada.

Se a *.dll* do CLP tiver sido excluída da pasta *../SOURCE* dentro do projeto ou o nome tivesse sido alterado ou por qualquer outro motivo a *.dll* não foi encontrada, a mensagem seria de erro.

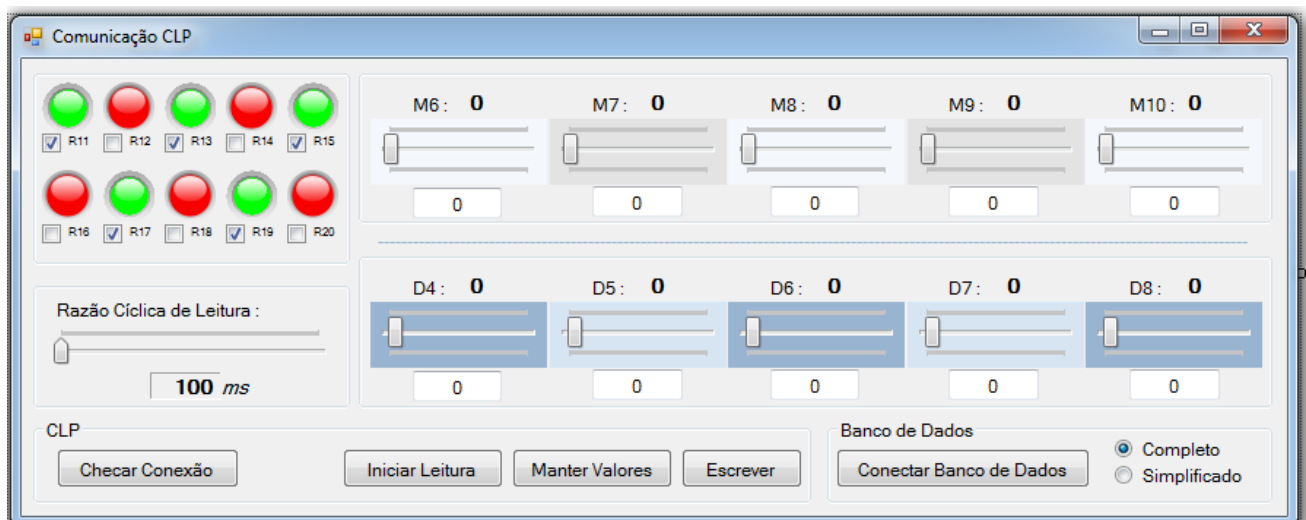


Figura 6. : Interface criada em C++.

Na figura 6 pode-se perceber a estrutura geral do programa de comunicação com esse CLP disponível em laboratório.

Explicando por partes cada função do software.

3- Software desenvolvido em C++

3-1. onLoad - ao carregar o programa

Ao carregar o programa a .dll vai ser importada via uma função em C++ chamada LoadLibrary. Caso ela tenha sido carregada com sucesso uma *messageBox* será mostrada para o usuário - o contrário também é verdade.

```
private: System::Void Form1_Load(System::Object^ sender, System::EventArgs^ e) {
    //Codigo de leitura da DLL
    Lib = LoadLibrary(L"../SOURCE/ScpHIV10.dll");
    if(Lib == NULL)
    {
        MessageBox::Show("Falha ao carregar a DLL");
        dll_carregada = false;
    }
    else{
        MessageBox::Show("DLL de Comunicação com CLP Carregada . . .");
        dll_carregada = true;
        configurar = (configCLP)GetProcAddress(Lib, "SCPConfigComChannel");
        check_CLP = (checarConexao)GetProcAddress(Lib, "SCPCheckConnection");
        abre_porta = (abrePorta)GetProcAddress(Lib, "SCPOpenPort");
        fecha_porta = (fechaPorta)GetProcAddress(Lib, "SCPClosePort");
        ler_dados = (lerDados)GetProcAddress(Lib, "SCPReadData");
        escrever_dados = (escreverDados)GetProcAddress(Lib, "SCPWriteData");
    }
}
```

Figura 7. : Código de carregamento da .dll.

3-2. Tempo de leitura cíclica

Logo em seguida o timer - ou o *clock* do programa - é definido e assim são chamadas várias funções todas as vezes que um pulso do timer for mandado. Essas funções são de *Leitura, Escrita no Banco de Dados e Mudança dos Estados dos Mostradores das Variáveis R, M e D*. Nota-se que aqui o timer será ativado pelo clicar no botão << *Iniciar Leitura* >>.

```
private: System::Void timer1_Tick(System::Object^ sender, System::EventArgs^ e) {
    leitura();
    addDataBD();
    statusLEDS(bufferR);
    statusM(bufferM);
    statusD(bufferD);
}
```

Figura 8. : Código de chamada de funções pelo timer.

Através de uma *trackBar* é possível escolher um valor que definirá o tempo de espera entre uma leitura e outra. Esse intervalo entre 100ms e 1000ms (valores padrões de leitura), mas pode ser modificado no código facilmente. O valor setado aparece em uma *textBox* logo abaixo da *trackBar*.

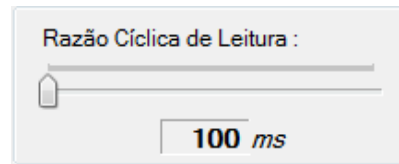


Figura 9. : Duração entre leituras.

O código que altera as propriedades do timer é o seguinte :

```
private: System::Void trackBar6_ValueChanged(System::Object^ sender, System::EventArgs^ e) {
    setTime();
    label_razao->Text = Convert::ToString(this->trackBar6->Value);
}
```

Figura 10. : Chamada da função setTime.

```
private: System::Void setTime(){
    this->timer1->Interval = Convert::ToInt32(this->trackBar6->Value);
}
```

Figura 11. : Modificação do timer.

Percebe-se aqui a preocupação em dividir o código em funções - mesmo que simples - afim de clarear a programação e auxiliar em uma melhor documentação do código.

Em uma parte do programa a *textBox* que mostra o tempo de leitura cíclica é alterada e em outra parte esse valor é recuperado afim de modificar o timer do programa.

3-1. Leitura dos dados do CLP

Os dados do CLP são lidos através de buffer declarados no início do programa :

- `unsigned char` bufferR[10];
- `short int` bufferM[5];
- `float` bufferD[5];

É importante notar também que essas variáveis são única e exclusivamente para ler os dados enviados pelo CLP. Outras variáveis referenciadas por *bufferR_esc*, *bufferM_esc* e *bufferD_esc* são para a escrita no dispositivo.

3-1.1. Função de mudança de estado para os LEDs

```
private: System::Void statusLEDs(unsigned char bufR[10]) {
    String^ nome_caminho = "../SOURCE/IMGS/red-led.png";
    bool boolean_value;
    for (int i = 0; i < 10; i++) {
        if (bufR[i] == 0) {
            // LED VERMELHO
            boolean_value = false;
            nome_caminho = "../SOURCE/IMGS/red-led.png";
        }
        else {
            // LED VERDE
            boolean_value = true;
            nome_caminho = "../SOURCE/IMGS/green-led.png";
        }
        if (i == 0){
            leds_vec[0] = boolean_value;
            ledR11->Load(nome_caminho);
        }
        if (i == 1){
            leds_vec[1] = boolean_value;
            ledR12->Load(nome_caminho);
        }
    }
}
```

Figura 12. : Código de mudança dos status dos LEDs.

Explicado brevemente, o programa varre todos os valores dentro do bufferR e, de acordo com os valores booleanos de cada índice, ele define uma imagem padrão (ou LED vermelho para 0 ou LED verde para 1) e muda essas imagens das *labels* correspondentes - de R11 à R20.

3-1.2. Função de mudança de estado para as variáveis M

Para as variáveis M o código é bastante simples. Pega-se os valores lidos no bufferM e escreve nas labels e trackBars correspondentes - de M6 à M10.

```
private: System::Void statusM(short int bufM[5]){
    this->m6tbar->Value = Convert::ToInt16(bufM[0]);
    this->m6val->Text = Convert::ToString(bufM[0]);
    this->m7tbar->Value = Convert::ToInt16(bufM[1]);
    this->m7val->Text = Convert::ToString(bufM[1]);
    this->m8tbar->Value = Convert::ToInt16(bufM[2]);
    this->m8val->Text = Convert::ToString(bufM[2]);
    this->m9tbar->Value = Convert::ToInt16(bufM[3]);
    this->m9val->Text = Convert::ToString(bufM[3]);
    this->m10tbar->Value = Convert::ToInt16(bufM[4]);
    this->m10val->Text = Convert::ToString(bufM[4]);
}
```

Figura 13. : Código de mudança dos valores das variáveis M.

3-1.3. Função de mudança de estado para as variáveis D

E para D é análogo, apenas com um pequeno detalhe : foi preciso usar uma constante multiplicativa (no caso 10) para ajustar o valor lido em D para um valor superior possível de ser convertido em String sem perder informações.

3-1. Escrita de dados

Antes de escrever os dados no CLP foi criada uma função para manter os dados lidos. Isso evita que dados não modificados sejam deletados, *resetados* pela gravação.

Assim, todos os valores lidos são transferidos para *textBoxes* auxiliares e *checkButtons*.

Ao clicar em << *Escrever Dados* >> o programa recupera os valores escritos nesses elementos de interface citados anteriormente e enviar, através de um buffer correto e uma função correspondente, ao CLP da seguinte maneira :

```
private: System::Void btnEscrever_Click(System::Object^ sender, System::EventArgs^ e) {
    unsigned char bufferR_esc[10] = {0,0,0,0,0,0,0,0,0,0};
    short int bufferM_esc[5];
    float bufferD_esc[5];
    int b;
    b = abre_porta("../SOURCE/ScpHi.cfg");
    if(this->cb11->Checked)
        bufferR_esc[0] = 255;
    if(this->cb12->Checked)
        bufferR_esc[1] = 255;
    if(this->cb13->Checked)
        bufferR_esc[2] = 255;
    if(this->cb14->Checked)
        bufferR_esc[3] = 255;
    if(this->cb15->Checked)
        bufferR_esc[4] = 255;
    if(this->cb16->Checked)
        bufferR_esc[5] = 255;
```

Figura 14. : Inicialização de buffers temporários de escrita e recuperação de valores da interface.

3-2. Banco de dados

A figura abaixo mostra à região de seleção do usuário relacionada ao banco de dados :

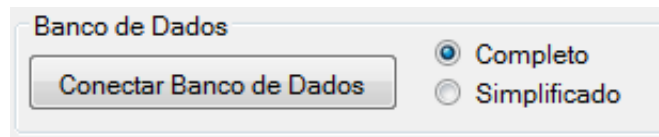


Figura 15. : Banco de dados na interface gráfica.

```
private: System::Void conectarBD_Click(System::Object^ sender, System::EventArgs^ e) {
    conexao= PQconnectdb("host=localhost user=postgres password=redes dbname=redes");
    if (PQstatus(conexao) != CONNECTION_OK)
    {
        PQfinish(conexao);
        return;
    }
}
```

Figura 16. : Comunicação com o banco de dados.

Bancos de dados ou bases de dados são coleções organizadas de informações (dados) que se relacionam de forma a criar um sentido e dar mais eficiência durante uma pesquisa ou estudo. São de vital importância para empresas, e há duas décadas se tornaram a principal peça dos sistemas de informação.

O banco de dados desse programa foi desenvolvido em SQL com auxílio do software PostgreSQL.

Para manipular tabelas em SQL é preciso importar a biblioteca *libpq-fe* dentro do projeto, a qual irá procurar vários outros arquivos auxiliares dentro do repertório do projeto.

Assim, com essa biblioteca carregada, é possível utilizar as funções SQL facilmente no programa desenvolvido em C++ :

```
private: System::Void addDataBD(){
    if (!conexao || PQstatus(conexao) != CONNECTION_OK)
    {
        return;
    }

    if(leds_vec[0] != bool(bufferR[0]) || this->completoBD->Checked){
        sprintf_s(nome_var_sql,"INSERT INTO \"clp\"(\"nome\", \"valor\", \"tempo\") VALUES('%s',%d,'now')",\"r11\",(int)bufferR[0]);
        result = PQexec(conexao, nome_var_sql);
    }
    if(leds_vec[1] != bool(bufferR[1]) || this->completoBD->Checked){
        sprintf_s(nome_var_sql,"INSERT INTO \"clp\"(\"nome\", \"valor\", \"tempo\") VALUES('%s',%d,'now')",\"r12\",(int)bufferR[1]);
        result = PQexec(conexao, nome_var_sql);
    }
    if(leds_vec[2] != bool(bufferR[2]) || this->completoBD->Checked){
        sprintf_s(nome_var_sql,"INSERT INTO \"clp\"(\"nome\", \"valor\", \"tempo\") VALUES('%s',%d,'now')",\"r13\",(int)bufferR[2]);
        result = PQexec(conexao, nome_var_sql);
    }
    if(leds_vec[3] != bool(bufferR[3]) || this->completoBD->Checked){
        sprintf_s(nome_var_sql,"INSERT INTO \"clp\"(\"nome\", \"valor\", \"tempo\") VALUES('%s',%d,'now')",\"r14\",(int)bufferR[3]);
        result = PQexec(conexao, nome_var_sql);
    }
}
```

Figura 17. : Escrevendo dados no banco de dados.

Essa parte do programa só escreve os valores lidos do CLP em um banco de dados. A parte de aquisição termina aqui. O próximo passo é possibilitar o acesso remoto.

Configuração do Servidor

Para essa parte de configuração de um servidor, criação de uma página web foram utilizadas as seguintes ferramentas :

- *PostgreSQL 9.3 (versão 64 bits)* - para a criação e visualização do banco de dados ;
- *Apache 2.2.22 (versão 32 bits)* - para a configuração do servidor em localhost;
- *PHP 5.5 (versão 32 bits)* - para criação de páginas *HTML* dinâmicas ;
- *Adobe Photoshop (versão 64 bits)* - para a criação do design/template da página ;

1- Desenvolvimento do banco de dados

Após a instalação do PostgreSQL, foi criado um banco de dados chamado *redes*. Dentro desse banco de dados foi criada uma tabela chamada *clp*. E dentro dessa tabela 3 colunas de variáveis : nome, valor e tempo.

A coluna nome armazena os tipos e as posições das variáveis (ex. R11, M7, D5, etc...). A coluna valor armazena o valor relacionado àquela variável (ex. se R11=true, valor armazena 255). A coluna tempo armazena a data e hora exatas de leitura de cada variável.

A tabela fica assim :

	nome character varying(3)	valor real	tempo timestamp without time zone
1	r16	0	2013-12-02 13:13:30.771
2	r11	255	2013-12-02 13:17:14.438
3	r13	0	2013-12-02 14:00:05.942
4	d4	32	2013-12-02 19:38:33.969
5	d4	32	2013-12-02 19:38:44.293
6	d7	22	2013-12-02 21:06:08.904
7	r11	255	2013-12-03 12:12:59.645
8	r12	255	2013-12-03 12:12:59.754
9	r13	255	2013-12-03 12:12:59.755
10	r14	255	2013-12-03 12:12:59.756
11	r15	255	2013-12-03 12:12:59.756
12	r16	255	2013-12-03 12:12:59.757

Figura 18. : Exemplificação da tabela criada com alguns valores já escritos.

É importante deixar claro que essas linhas são escritas no programa em C++, como foi mostrado anteriormente.

2- Criação de uma página web

A página criada é mostrada na figura abaixo :

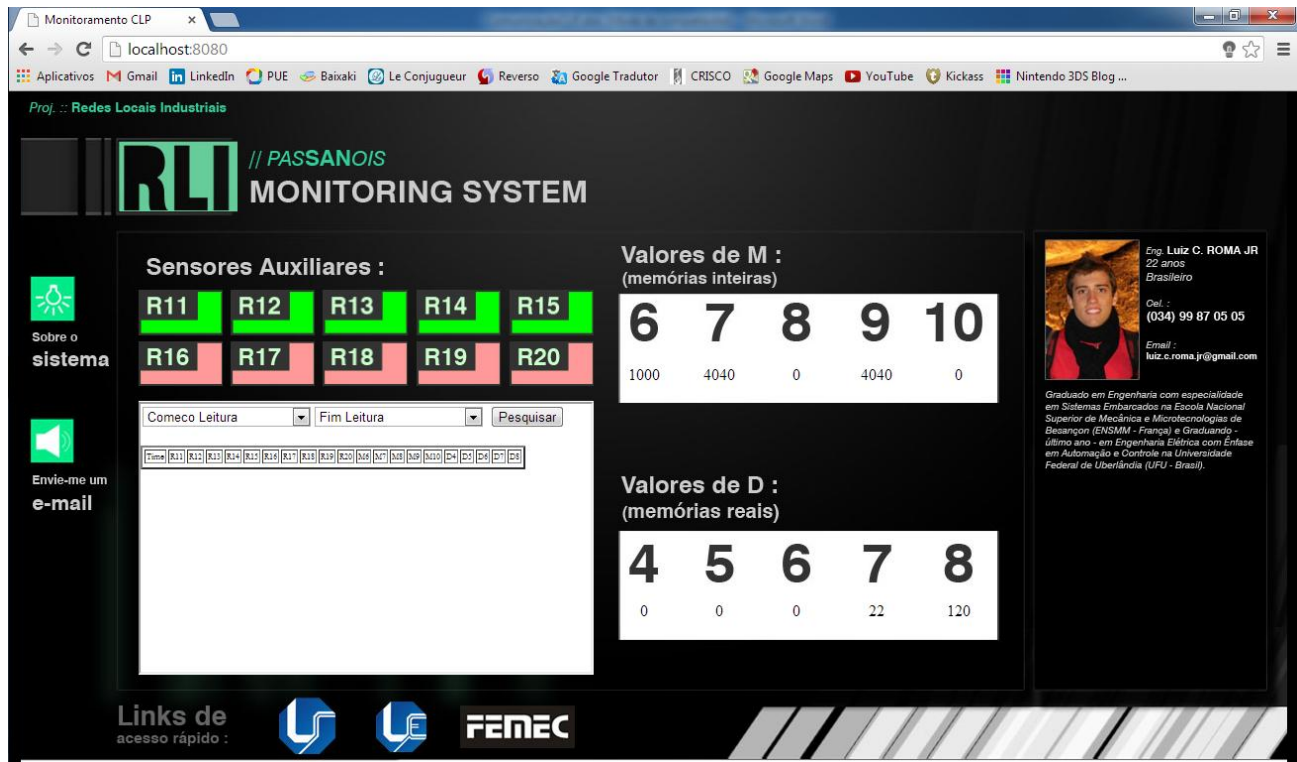


Figura 19. : Página web desenvolvida em PHP + Photoshop + HTML + SQL.

2-1. Criando página web no Photoshop

O design da interface gráfica foi feito totalmente no Adobe Photoshop versão CS5. Nesse software é possível desenhar e criar imagens como o usuário bem entender e, através de uma ferramenta de corte, defini-se regiões/áreas interessantes para um código HTML.

Por trás, o Photoshop trabalha na idéia de tabelas. Assim, quanto melhor for a ideia de particionamento de imagens em células de uma tabela, mais fácil de criar uma interface e melhor o código será.

A seguir pode-se perceber uma divisão do *template* criado em várias fatias. Essas fatias serão celular em uma tabela HTML :

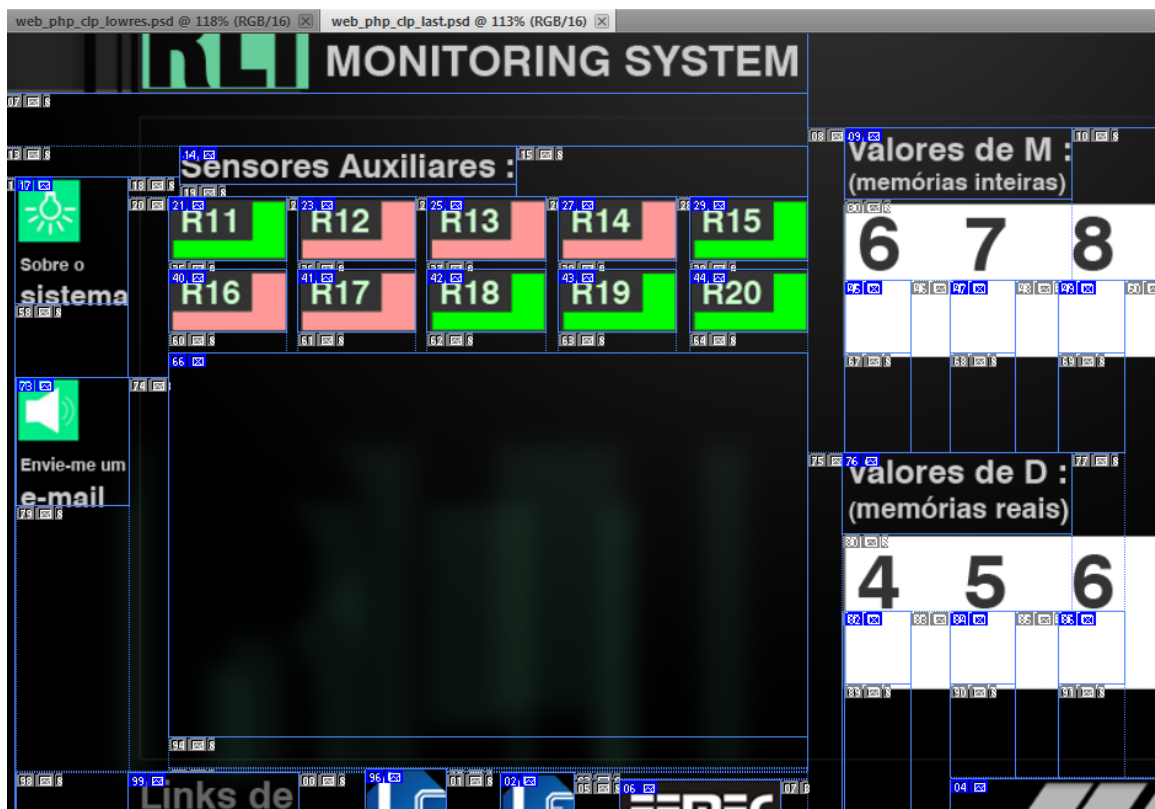


Figura 20. : Exemplo de recorte da imagem em células.

Existe um modo de salvar o projeto feito em Photoshop em forma de página Web. Nessa opção, o programa cria um código HTML correspondente às fatias criadas no projeto. Com esse código pronto, pode-se passar para a próxima etapa : edição PHP. À seguir, parte do código criado pelo Photoshop para a página Web desenvolvida :

```
<tr>
  <td colspan="2" rowspan="19">
    
  <td colspan="5" rowspan="5">
    
  <td colspan="6" rowspan="10">
    
  <td rowspan="14">
    <a href="http://buscatextual.cnpq.br/buscatextual/visualizacv.do?id=K4466610U7" target="blank2">
      
    <td colspan="4" rowspan="6">
      
    <td>
      
  </td>
</tr>
<tr>
  <td colspan="6">
    
  <td colspan="10" rowspan="2">
    
  <td colspan="8" rowspan="3">
    
  <td>
    
  </td>
</tr>
```

Figura 21. : Estrutura do código HTML criado pelo Photoshop.

2-2. Implementando PHP no código HTML existente

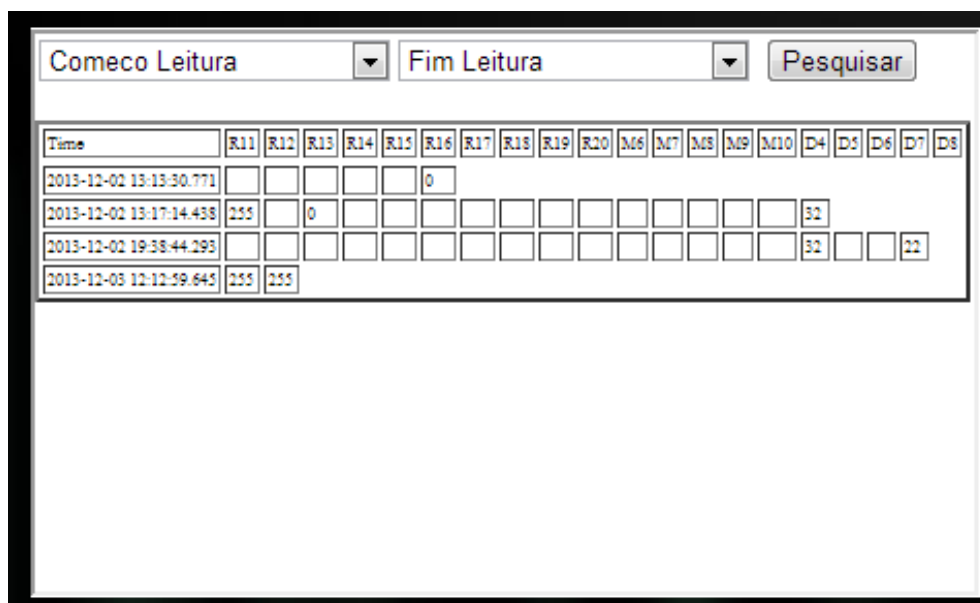
Após ter feito teste em códigos PHP básicos, testado como a linguagem funciona, algumas implementações PHP foram feitas pelas linhas HTML existentes.

Uma coisa à citar é que o *website* de monitoramento foi dividido em 2 páginas. A página principal - tela preta inicial - é a mainframe, onde há informações úteis sobre o desenvolvimento e atualização dos valores de acordo com as últimas atualizações feitas no CLP. Pode ser considerada a parte Tempo Real da página.

O pequeno quadrado em branco onde fica << *Comeco Leitura* >> e << *Fim Leitura* >> é uma segunda página que implementa uma tabela dinâmica criada à partir de dados escritos no banco de dados.

As imagens de R11 à R20 mudam de acordo com o *AutoRefresh* em javascript presente na página inicial, assim como os valores de M6 à M10 e de D4 à D8.

A parte mais complicada do código foi a criação da tabela dinâmica :



Time	R11	R12	R13	R14	R15	R16	R17	R18	R19	R20	M6	M7	M8	M9	M10	D4	D5	D6	D7	D8
2013-12-02 13:13:30.771						0														
2013-12-02 13:17:14.438	255		0														32			
2013-12-02 19:38:44.293																	32			22
2013-12-03 12:12:59.645	255	255																		

Figura 22. : Tabela dinâmica para filtrar dados ocorridos em um determinado período existente no banco de dados.

Para criar essa tabela, os dados selecionados foram filtrados do banco de dados em SQL por data e, posteriormente, por nome. Porém, como o programa C++ grava o horário com precisão em milissegundos, dados adquiridos em um mesmo momento são separados na hora de serem inseridos na tabela.

Assim, o primeiro código criado para escrever as linhas da tabela por ordem de tempo fazia com que valores que deveriam estar em uma mesma linha estivessem em linhas diferentes e a tabela era, então, composta por linhas com um único valor.

Para resolver esse problema foi necessário dividir a string de tempo em substrings e retirar 2 casas finais dela, afim homogeneizar o tempo entre variáveis que foram lidas no mesmo instante.

Conclusão

Este documento teve como objetivo a descrição geral da metodologia utilizada para a criação desse projeto de comunicação entre um CLP tipo industrial e um computador.

Foi um módulo de extrema importância, com problemas reais (problemas de compatibilidade entre arquiteturas de programas e sistemas operacionais; aprendizado de linguagens de programação nunca ou pouco utilizadas; revisão de conceitos importantes de programação; utilização de bibliotecas externas e dlls; etc).

Bibliografia

- [1] : http://www.php.net/manual/pt_BR/, acessado durante o módulo 1 inteiro.
- [2] : <http://www.apache.org/>, acessado na primeira aula de laboratório.
- [3] : http://pt.wikipedia.org/wiki/Banco_de_dados, acessado dia 07-12-2013.
- [4] : http://pt.wikipedia.org/wiki/Controlador_1%C3%B3gico_program%C3%A1vel, acessado dia 07-12-2013.
- [5] : 1º módulo prático - Rede Física e Lógica Cliente/Servidor, prof. José Jean-Paul Zanlucchi de Souza Tavares.
- [6] : DLL de Comunicação SCP-HI Versão 10, HI Tecnologia, setembro 2005, PNS.0016.
- [7] : DLL de Comunicação SCP-HI Versão 10, HI Tecnologia, setembro 2005, PNS.0016.
- [8] : <http://www.w3schools.com/>, acessado durante todo o módulo.
- [9] : <http://www.cplusplus.com/>, acessado durante todo o módulo.