Universidade Federal de Uberlândia
www.ufu.br

Práticas de laboratório para disciplina de 'Sistemas em Tempo Real'

Prof. Alan Petrônio Pinheiro

Faculdade de Engenharia Elétrica – www.feelt.ufu.br
Curso de graduação em Engenharia de Computação
alanptronio@ufu.br

Versão 1.0 – 2024

SUMÁRIO

Universidade Federal de Uberlândia

Prática 1: aprendendo a usar Threads

Prof. Alan Petrônio Pinheiro
Faculdade de Engenharia Elétrica - Curso de Engenharia de Computação
Versão destes prática: 1.1

---

1 - Objetivos desta prática

Introduzir o uso básico de threads em aplicações afim de conseguir melhor desempenho de execução e baixar tempos para melhorar o atendimento a requerimentos de tempo para sistemas STR.

Para isto, esta prática usa o exemplo prático de uma aplicação que varre um range de IPs na rede buscando, um por um, se estão ativos. Um aplicativo comercial que faz isto é o "Angry IP" (download aqui). Desejamos fazer algo semelhantes para ilustrar o uso de threads em sistemas STR. Sugere-se baixar este software e testar para entender melhor a aplicação e até mesmo avaliar o desempenho da sua aplicação com este programa de referência.

2 – Background: entendendo o básico de Threads para C# e .NET

Alguns pontos que o programador deve entender para proceder com a prática:

I. Na tecnologia .NET as "managed threading" são implementadas pela classe System.Threading.Thread
II. "Unmanaged threading" são as threads que são gerenciadas quando a programação usa elementos de thread nativas da plataforma Win32
III. É possível que unmanaged threads serem chamadas dentro de um processo "managed" (rodando em .NET) usando as tecnologias "COM interop" ou através da plataforma de chamadas PInvoke" de códigos .NET
IV. Uma thread pode ser unicamente identificada usando a propriedade "ManagedThreadId" da classe System.Threading.Thread. Ela retorna um inteiro que é garantido ser único durante todo o período de vigor da thread e não pode ser alterado depois de iniciado.
V. A propriedade "ThreadState" é somente de leitura e indica o estado da thread que pode ser:
    a. • Aborted: foi abortada.
    b. • AbortRequested: foi requisitado um aborto, mas ainda não foi executado.
    c. • Background: está sendo executada com baixa prioridade em plano de fundo (neste caso, a propriedade IsBackground tem que ser configurada para true)

d. • Running: está sendo atualmente sendo executada.
e. • Stopped: foi parada.
f. • StopRequested: foi solicitada para ser parada mais ainda está sendo concluída.
g. • Suspended: suspensa.
h. • SuspendRequested: foi solicitada para ser suspendida mais ainda está sendo concluída.
i. • Unstarted: foi criada, mas não foi iniciada.
j. • WaitSleepJoin: a thread esta bloqueada.

VI. A propriedade Thread.IsAlive que indica se a thread esta atualmente sendo executada ou não.

VII. Se você deseja saber a thread que está em execução, pode usar: `var currentThread = System.Threading.Thread.CurrentThread;`

VIII. Existem alguns tipos de exceção específicos para threading. Esses tipos de exceções geralmente podem ser evitados verificando a propriedade ThreadState antes de agir. As principais são:

a. ThreadInterruptedException,
b. ThreadAbortException (não é compatível com .NET 6),
c. ThreadStartException (se tem algum problema ao iniciar a execução)
d. ThreadStateException (lançada quando um método no thread é chamado e não é disponível). Iniciar em um thread que já foi iniciado é inválido e causará uma ThreadStateException

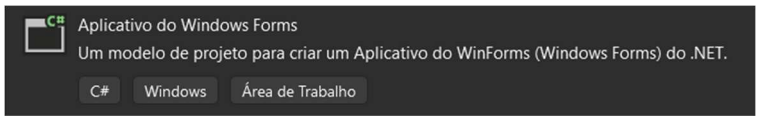
3 – Execução desta prática

Antes de proceder, destaca-se que o programa e seu código fonte acompanham este documento ou podem ser baixados neste link. Ele deve ser aberto em Visual Studio com C# instalado. Para reproduzir o projeto, basta seguir as seções na sequência.

3.1 – Criando o projeto e o básico da aplicação

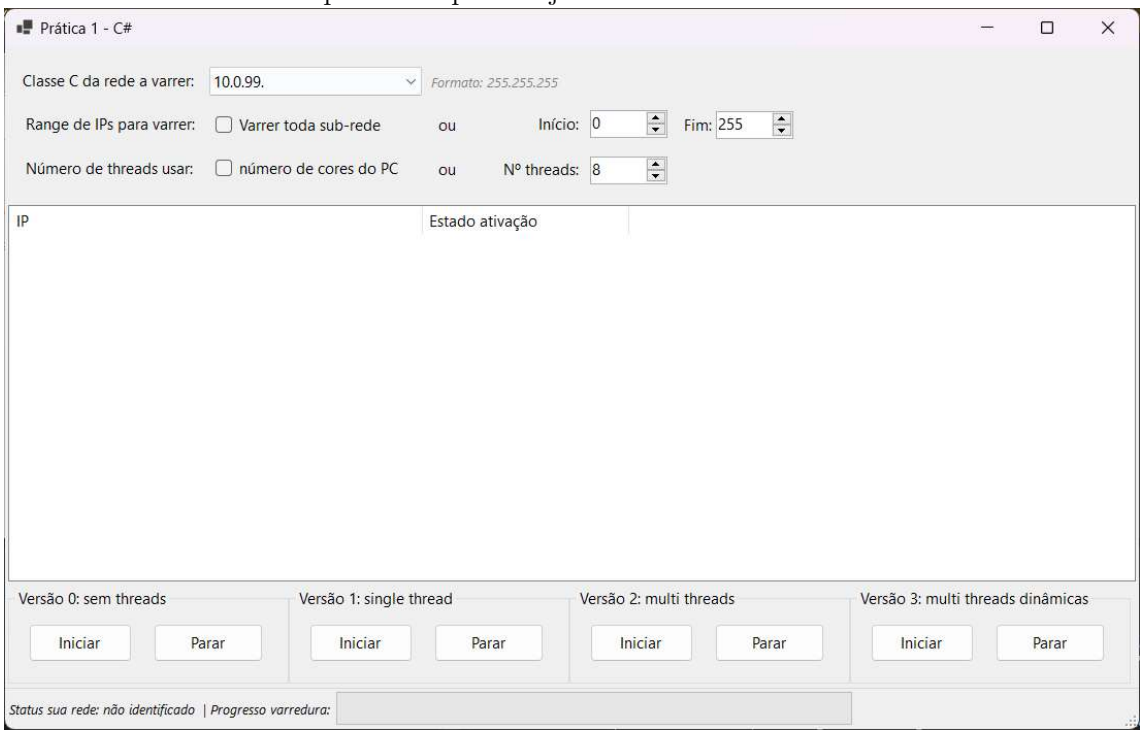
Siga a sequência de passos abaixo para criar um novo projeto no Visual Studio:

1) Crie um novo projeto de app C# para Windows usando a opção indicada na figura da sequência. Nomeie seu projeto como “Pratica1”.



2) Desenhe uma interface gráfica similar a que se vê na figura da sequência.
   a. Use primeiro o componente *tableLayoutPanel* para dimensionar bem a tela. Use 1 coluna, 4 linhas (150px, 100%, 100px, 20 px) e use a propriedade dock = fill.
   b. Nas linhas 1 e 3, coloque *Panel* com a propriedade dock=fill para receber os outros componentes.
   c. Na linha 2, coloque um *ListView*. Configura o atributo “View = details” e insira as colunas no atributo “Columns” conforme figura.

d. Na linha 4, coloque um statusStrip conforme figura.
e. Altere todas as propriedades da interface (Form) e dos componentes que desejar.



3.2 – Introduzindo os códigos básicos de eventos

Siga a sequência de passos abaixo para introduzir as partes que envolvem thread:

1) Antes de entrar no código propriamente dito, vamos a primeira convenção do código que é fundamental para você entendê-lo: o vetor "resultadoBusca". Ele é um vetor de inteiros, que funcionará como uma variável global para a classe principal do programa (a Form1) e todas as threads irão guardar o resultado de seu processamento nele. Como ele é global, será fará o papel de variável compartilhada entre as threads! Seu formato é mostrado na sequência para o exemplo onde vamos varrer a classe de IPs 10.3.192.45 a 10.3.192.50. Neste exemplo, os valores em azul representam o final do IP testado (ocupam as posições pares do vetor) e os valores em vermelho, o resultado da busca (ocupam as posições ímpares do vetor). O valor 1 indica que o IP respondeu. O valor 0, indica que o IP foi testado com um ping, mas não respondeu. E o valor -1 indique que o IP ainda sequer foi testado (não foi enviado pacote de ping). Cada thread, se responsabilizará por um grupo de IPs. Por isto, diferentes threads acessam esta mesma estrutura de dados, mas em porções específicas.

Index vetor i	0	1	2	3	4	5	6	7	8	9	10	11
Valores do vetor	45	1	46	0	47	-1	48	-1	49	0	50	1

2) Na classe principal, declare os objetos e atributos conforme visto no código abaixo da linha 4 a 10. Ainda, no construtor da classe, coloque o código da linha 15 a 32. Na linha 15, temos um importante comando da linguagem que faz com que o SO desligue seu mecanismo de verificação que proíbe que diferentes threads acessem um mesmo recurso ao mesmo tempo. No caso, nós permitiremos que as threads possam compartilhar recursos entre si. Mas tomaremos o cuidado devido para evitar problemas.

```
[1]     public partial class Form1 : Form
[2]     {
[3]
[4]         private Thread threadBuscadora; //thread unica usada para varrer IPs
[5]         Thread[] listaThreads;
[6]         private Stopwatch marcadorTempoExecucao; //faz contagem tempo execucao codigo
[7]         private int quantidadeTotalIPsVarrer = 0;  //recebe da thread qtd ips varridos. Por isto eh
[8]         public int inicioVarredura = 0;
[9]         public int fimVarredura = 0;
[10]        public int[] resultadoBusca; //resultados sao guardados aqui. Posicao pares é o final do ip..
[11]
[12]        public Form1()
[13]        {
[14]            InitializeComponent();
[15]            Control.CheckForIllegalCrossThreadCalls = false; //possibilita que componentes sejam chamados
[16]            var host = Dns.GetHostEntry(Dns.GetHostName());
[17]            string classe_rede_minha_interface_rede = string.Empty;
[18]            try
[19]            {
[20]                foreach (var ip in host.AddressList)
[21]                    if (ip.AddressFamily == AddressFamily.InterNetwork)
[22]                    {
[23]                        classe_rede_minha_interface_rede = ip.ToString().Split('.')[0] + "." +
[24] ip.ToString().Split('.')[1] + "." + ip.ToString().Split('.')[2] + ".";
[25]                        comboBoxIPsDisponiveis.Items.Add(classe_rede_minha_interface_rede);
[26]                    }
[27]                comboBoxIPsDisponiveis.SelectedIndex = 0; //SELECIONA primeiro IP aparecer
[28]            }
[29]            catch (Exception ex)
[30]            {
[31]                MessageBox.Show("Não pode encontrar IP deste dispositivo: " + ex.Message.ToString());
[32]            }
[33]
[34]        }
[35] ......
```

3) Vamos agora criar o método que varre IPs. Ele é mostrado no código da sequência. Através de um comando de PING (uma aplicação que verifica se existe um dispositivo usando aquele IP na rede) será feira a varredura da rede.

```
[1]       public void VarreIPs() //rotina que faz a varredura de IPs. Eh a rotina de thread
[2]       {
[3]           string classe = comboBoxIPsDisponiveis.Text;
[4]           string ipParaPingar = string.Empty;
[5]
[6]           Ping ping = new Ping();
[7]           PingReply resposta;
[8]
[9]           int posicao_varredura_vetor = 1;
[10]          for (int i = inicioVarredura; i <= fimVarredura; i++)
[11]          {
[12]              ipParaPingar = classe + i.ToString();
[13]              try
[14]              {
[15]                  resposta = ping.Send(ipParaPingar);
[16]                  if (resposta.Status == IPStatus.Success)
[17]                      resultadoBusca[posicao_varredura_vetor] = 1; //achou
[18]                  else
[19]                      resultadoBusca[posicao_varredura_vetor] = 0; //nao achou
[20]                  posicao_varredura_vetor = posicao_varredura_vetor + 2; //varre de 2 em 2
[21]              }
[22]              catch (Exception ex)
[23]              {
[24]                  MessageBox.Show($"Ocorreu um erro ao fazer o ping: {ex.Message}");
[25]                  return;  //deixa de varrer
[26]              }
[27]          }
[28]      }
```

4) Vamos criar também a rotina de inicializações que são comuns a todas as versões de varredura mostradas na interface gráfica. Ele se encarrega de fazer inicializações da estrutura global de dados, quantidade de IPs e início da contagem de tempo. Segue função:

```
[1]         private void RotinaComumIniciarVarredura()
[2]         {
[3]             listViewResultadoIPs.Items.Clear(); //apaga escaneamentos anteriores
[4]             toolStripProgressBarVarredura.Value = 0; //inicia barra progresso
[5]
[6]             if (checkBoxTodaSubrede.Checked) //chegar quantos IPs varrer
[7]             {
[8]                 inicioVarredura = 0;
[9]                 fimVarredura = 255;
[10]            }
[11]            else
[12]            {
[13]                inicioVarredura = (int)(numericUpDownInicio.Value);
[14]                fimVarredura = (int)(numericUpDownFim.Value);
[15]
[16]            }
[17]            quantidadeTotalIPsVarrer = fimVarredura - inicioVarredura + 1; ;
[18]
[19]            toolStripProgressBarVarredura.Maximum = quantidadeTotalIPsVarrer; //configurando passos barra
[20] progresso
[21]
[22]            //inicializa estrutura de dados com resultados
[23]            resultadoBusca = new int[2 * quantidadeTotalIPsVarrer]; //Padrao: { 0, -1, 1, -1, 2, -1, 3, -1
[24] ....};
[25]            int contador = inicioVarredura;
[26]            for (int i = 0; i < 2 * quantidadeTotalIPsVarrer;)
[27]            {
[28]                resultadoBusca[i] = contador;
[29]                contador++;
[30]                resultadoBusca[i + 1] = -1;
[31]                i = i + 2;
[32]            }
[33]
[34]            marcadorTempoExecucao = new Stopwatch(); //classe  contabiliza tempo de execucao
[35]            marcadorTempoExecucao.Start(); //inicia contagem de tempo
[36]            timerExibicaoResultados.Start();
[37]        }
```

5) Para fins de visualização empregamos uma estratégia onde usamos um timer que atualização a 'list view' a cada 1 segundo, quando o processo de varredura começa. Para fins de simplificação, a cada 1 segundo ela limpa o componente gráfico, varre a lista de busca com contém os resultados e as posições de resultados que tem valor 0 ou 1 são plotadas. A interface gráfica desta aplicação é atualizada por um timer. Por isto, você deve criar um timer na aplicação principal. Atribua a propriedade .INTERVAL=1000 (faz ele estourar a cada 1 segundo) e ative o evento "TICK" dele. Neste evento, insira o código abaixo:

```
[1]         private void timerExibicaoResultados_Tick(object sender, EventArgs e)
[2]         {
[3]             //limpa a lista de visualizacao
[4]             listViewResultadoIPs.Items.Clear();
[5]             toolStripStatusLabelTempo.Text = "Executado varredura... (" +
[6] Convert.ToString(marcadorTempoExecucao.ElapsedMilliseconds / 1000) + " segs)";
[7]
[8]
[9]             int resultadosJaEncontrados = 0;
[10]            for (int i = 0; i < 2 * quantidadeTotalIPsVarrer;) //comeca acessando primeiro resultado
[11]            {
[12]                if (resultadoBusca[i + 1] == 1)
[13]                {
[14]                    resultadosJaEncontrados++;
[15]                    listViewResultadoIPs.Items.Add(new ListViewItem(new String[] { textBoxCampoIP.Text +
[16] resultadoBusca[i].ToString(), "Ligado" }));
[17]                }
[18]                if (resultadoBusca[i + 1] == 0)
[19]                {
```

```
[20]                    resultadosJaEncontrados++;
[21]                    listViewResultadoIPs.Items.Add(new ListViewItem(new String[] { textBoxCampoIP.Text +
[22] resultadoBusca[i].ToString(), "---" }));
[23]                }
[24]                i = i + 2;
[25]            }
[26]
[27]
[28]            //verifica se ja processou tudo. Se processou, para este timer
[29]            toolStripProgressBarVarredura.Value = resultadosJaEncontrados;
[30]            if (resultadosJaEncontrados >= quantidadeTotalIPsVarrer)
[31]            {
[32]                marcadorTempoExecucao.Stop();
[33]                timerExibicaoResultados.Stop();
[34]                toolStripProgressBarVarredura.Value = 0;
[35]                toolStripStatusLabelTempo.Text = "Concluído em: " +
[36] Convert.ToString(marcadorTempoExecucao.ElapsedMilliseconds / 1000) + " seg. Timer fechado";
[37]            }
[38]        }
```

6) No evento 'CLICK' do botão de aplicação 'sem thread', inclua o código abaixo. Evite colocar um range grande de IPs para não travar a aplicação.

```
[1]         private void buttonIniciar_Click(object sender, EventArgs e) //sem threads
[2]         {
[3]             RotinaComumIniciarVarredura(); //algums funcoes secundarias de inicializacao interface
[4]             VarreIPs();
[5]         }
```

3.3 – Usando threads: versão 1 com single thread

Nestes passos, usamos 1 thread de forma mais básica possível. Para isto, faça:

1) No evento 'CLICK' do botão de aplicação 'single thread', inclua o código abaixo.

```
[1]         private void buttonSingleThread_Click(object sender, EventArgs e) //com 1 thread simples
[2]         {
[3]             RotinaComumIniciarVarredura(); //algums funcoes secundarias de inicializacao interface
[4]             threadBuscadora = new Thread(new ThreadStart(VarreIPs));
[5]             threadBuscadora.Name = "Rastreador de rede";
[6]             threadBuscadora.Start();
[7]             //threadBuscadora.Join(); //enquanto thread nao finalizar, para aqui
[8]         }
```

2) Execute o código ora habilitando o método "JOIN", ora desabilitando. Veja a diferença entre as execuções.

3.4 – Usando threads: versão 2 com multithreads criadas estaticamente

Para usar mais do que 1 thread, precisamos que as n threads compartilhem a região de memória de resultados representada pela variável 'ResultadoDaBusa[int]' :

1) Será necessário criar uma nova classe para varrer IPs uma vez que agora precisaremos passar valores para as Threads. Uma forma de fazer isto é criando uma classe, dentro do mesmo *namespace*, conforme visto no código abaixo. Observe também a existência de um MUTEX para bloquear a região de código onde se acessa a estrutura de dados (memória) compartilhada entre threads.

```
[1] public class VarreduraIPs
```

```
[2]  {
[3]      public int primeiroIPVarrer = 0; //informacao usada para acessar estrutura de dados
[4]      public int inicio = 0;
[5]      public int fim = 0;
[6]      public string classe = string.Empty;
[7]      public int[] resultadoBusca;
[8]      private static Mutex nossoMutex = new Mutex();
[9]
[10]     public VarreduraIPs(string classeC_IP, int inicio, int fim, int[] copia_resultadoBusca, int
[11] copia_primeiroIPVarrer)
[12]     {
[13]         this.inicio = inicio;
[14]         this.fim = fim;
[15]         this.classe = classeC_IP;
[16]         this.resultadoBusca = copia_resultadoBusca;
[17]         this.primeiroIPVarrer = copia_primeiroIPVarrer;
[18]     }
[19]     public void VarreIPsMelhorada() //rotina que faz a varredura de IPs. Eh a rotina de thread
[20]     {
[21]         string ipParaPingar = string.Empty;
[22]
[23]         Ping ping = new Ping();
[24]         PingReply resposta;
[25]
[26]         int posicao_varredura_vetor;
[27]         for (int i = inicio; i <= fim; i++)
[28]         {
[29]             ipParaPingar = classe + i.ToString();
[30]             try
[31]             {
[32]                 resposta = ping.Send(ipParaPingar);
[33]                 posicao_varredura_vetor = (i - primeiroIPVarrer) * 2;
[34]                 nossoMutex.WaitOne(); //bloqueia esta regiao para 1 simples thread acessar
[35]                 if (resposta.Status == IPStatus.Success)
[36]                     resultadoBusca[posicao_varredura_vetor + 1] = 1; //achou
[37]                 else
[38]                     resultadoBusca[posicao_varredura_vetor + 1] = 0; //nao achou
[39]                 nossoMutex.ReleaseMutex(); //desbloqueia esta regiao
[40]                 posicao_varredura_vetor = posicao_varredura_vetor + 2; //varre de 2 em 2
[41]             }
[42]             catch (Exception ex)
[43]             {
[44]                 MessageBox.Show($"Ocorreu um erro ao fazer o ping: {ex.Message}");
[45]                 return;  //deixa de varrer
[46]             }
[47]
[48]         }
[49]     }
[50] }
```

2) 'CLICK' do botão de aplicação 'multiplas thread estáticas', inclua o código abaixo. Observe que criamos 2 threads e dividimos o trabalho entre elas pela metade. A primeira thread varre a primeira metade de range de IPs e a segunda metade varre a outra metade. Observe agora que os valores passados pelas threads vão nos atributos criados pela nova classe exclusiva para varredura de IPs.

```
[51]         private void buttonMultiThread_Click(object sender, EventArgs e)
[52]         {
[53]             RotinaComumIniciarVarredura();
[54]             int metade = (int)(quantidadeTotalIPsVarrer / 2);
[55]             int inicio_primeira_metade = inicioVarredura;
[56]             int fim_primeira_metade = metade;
[57]             int inicio_segunda_metade = fim_primeira_metade + 1;
[58]             int fim_segunda_metade = fimVarredura;
[59]
[60]             Pratica1.VarreduraIPs objetoParaVarredura1 = new VarreduraIPs(textBoxCampoIP.Text,
[61] inicioVarredura, fim_primeira_metade, resultadoBusca);
[62]             Pratica1.VarreduraIPs objetoParaVarredura2 = new VarreduraIPs(textBoxCampoIP.Text,
[63] inicio_segunda_metade, fim_segunda_metade, resultadoBusca);
[64]
[65]             Thread t1 = new Thread(new ThreadStart(objetoParaVarredura1.VarreIPsMelhorada));
[66]             Thread t2 = new Thread(new ThreadStart(objetoParaVarredura2.VarreIPsMelhorada));
[67]             t1.Start();
[68]             t2.Start();
[69]             //t2.Join();
[70]             //t1.Join();
[71]         }
```

3.5 – Usando threads: versão 3 com multithreads criadas dinamicamente

Neste caso, ao contrário do anterior, vamos criar dinâmica o número de threads. Para isto, vamos ler do sistema operacional o número de cores do processador e dar para cada core, uma thread (em teoria uma vez que a atribuição de threads a cores é feita pelo SO). Para isto, faça:

1) No botão de iniciar multithreads, inclua o código na sequência. Observe que foi necessário criar um vetor de objetos que serão associados as threads que também serão armazenadas em um vetor de threads para que possam ser manipuladas.

```
[1]         private void buttonThreadsDinamicas_Click(object sender, EventArgs e)
[2]         {
[3]             RotinaComumIniciarVarredura();
[4]
[5]             int numCores;
[6]             if (checkBoxNumCores.Checked)
[7]                 numCores = Environment.ProcessorCount; //pega qtde 2*nucleos fisicos
[8]             else
[9]                 numCores = (int)(numericUpDownNumCores.Value);
[10]
[11]            listaThreads = new Thread[numCores];
[12]            VarreduraIPs[] objetosParaVarreduras = new VarreduraIPs[numCores];
[13]
[14]            int numIPsVarrerPorThread = (int)(quantidadeTotalIPsVarrer / numCores);
[15]            int inicio_Range_IPs = 0;
[16]            int fim_Range_IPs = 0;
[17]            if (numCores < 2)
[18]            {
[19]                MessageBox.Show("Esta função é para múltiplos CORES");
[20]                return;
[21]            }
[22]            for (int i = 0; i < numCores; i++) //distribuir range de IPs iguais para os primeiros n-1
[23] cores
[24]            {
[25]                //estima o range de IPs para distribuir entre as threads. A ultima vai pegar o "resto
[26]                if (i != (numCores - 1))  //nas primeiras threads, o range de IP eh igual
[27]                {
[28]                    inicio_Range_IPs = i * numIPsVarrerPorThread + inicioVarredura;
[29]                    fim_Range_IPs = inicio_Range_IPs + (numIPsVarrerPorThread - 1);
[30]                }
[31]                else //na ultima thread, vou jogar um bloco de ip numIPsVarrerPorThread + o resto
[32]                {
[33]                    inicio_Range_IPs = fim_Range_IPs + 1;
[34]                    fim_Range_IPs = fimVarredura;
[35]                }
[36]                objetosParaVarreduras[i] = new VarreduraIPs(comboBoxIPsDisponiveis.Text, inicio_Range_IPs,
[37] fim_Range_IPs, resultadoBusca, inicioVarredura);
[38]                listaThreads[i] = new Thread(new ThreadStart(objetosParaVarreduras[i].VarreIPsMelhorada));
[39]                listaThreads[i].Name = "Minha thread " + i.ToString();
[40]                listaThreads[i].Start();
[41]            }
            }
```

4 – Verificação da aprendizagem e observações relevantes

Para concluir, veja esta lista questionamentos e verifique se você é capaz de respondê-los ou se entendeu perfeitamente sua natureza ilustrativa.

a) Por que você acha que seu programa ficou aparentemente travado quando não usava thread?
b) Faça a função de parar varredura quando se está usando thread. E quando não se está, como parar?

c) Usando os comandos abaixo, crie uma outra thread que monitora se sua interface de rede está ativa ou não. Se ela desativar durante o scan, pare as threads de escaneamento.

```
[1]  var bgThread = new Thread(() =>
[2]     {
[3]        while (true)
[4]        {
[5]             bool redeLigada = System.Net.NetworkInformation.NetworkInterface.GetIsNetworkAvailable();
[6]             Thread.Sleep(500);
[7]        }
[8]     }
[9]                           );
[10]
[11] bgThread.IsBackground = true;
[12] bgThread.Start();
```

d) Como poderia ser feita uma função para levantamento de speedup desta aplicação?