Chat Multiusuário com Sockets e WinForms em C#

1. Introdução: Sockets, Threads e Invoke em WinForms

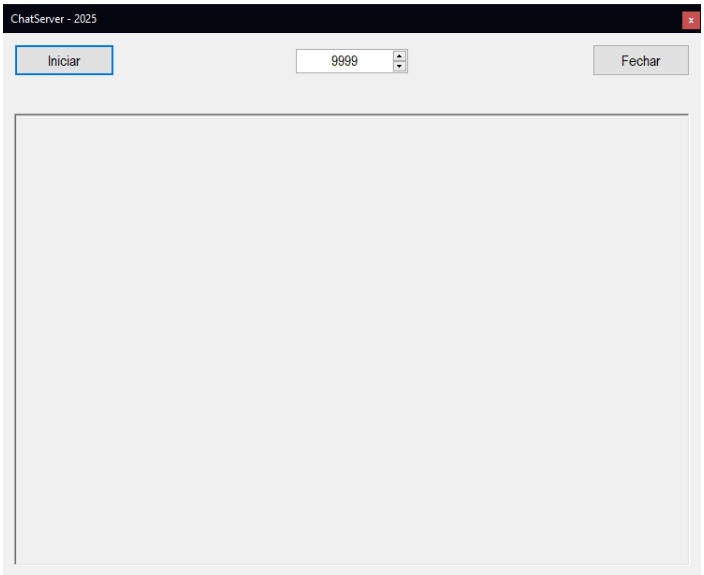
Em aplicações de sistemas distribuídos, sockets são usados para permitir comunicação entre processos em diferentes computadores. Em aplicações gráficas (WinForms), cada controle da interface deve ser manipulado apenas pela thread principal da interface. Portanto, ao usar threads para comunicação em rede (como em um chat), é obrigatório usar o método Invoke para atualizar a interface a partir dessas threads.

2. Estrutura da Solução

A solução é composta por dois programas: um servidor e um cliente, ambos desenvolvidos em WinForms. O servidor aceita múltiplos clientes e transmite todas as mensagens recebidas para todos os clientes conectados. O cliente permite ao usuário conectar, enviar e receber mensagens por meio de uma interface gráfica.

3. Servidor WinForms

O servidor deve conter:
- Um botão para iniciar o servidor;
- Um campo para porta;
- Uma área (RichTextBox) para exibir mensagens recebidas.



O servidor aceita conexões e retransmite mensagens para todos os clientes conectados.

Exemplo de código para o formulário principal do servidor:

```
using System;
using System.Collections.Generic;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Threading;
using System.Windows.Forms;

public partial class ServidorForm : Form
{
   TcpListener servidor;
   List<TcpClient> clientes = new List<TcpClient>();
   Thread threadServidor;
   bool ativo = false;

   public ServidorForm()
   {
      InitializeComponent();
   }

   private void buttonIniciar_Click(object sender, EventArgs e)
   {
      int porta = int.Parse(textBoxPorta.Text);
      servidor = new TcpListener(IPAddress.Any, porta);
      servidor.Start();
      ativo = true;
      threadServidor = new Thread(AceitarClientes);
      threadServidor.IsBackground = true;
      threadServidor.Start();
      richTextBoxLog.AppendText("Servidor iniciado.");
   }

   void AceitarClientes()
   {
      while (ativo)
      {
         TcpClient cliente = servidor.AcceptTcpClient();
         lock (clientes) clientes.Add(cliente);
         Thread t = new Thread(() => ReceberMensagens(cliente));
         t.IsBackground = true;
         t.Start();
```

```
    }
  }

  void ReceberMensagens(TcpClient cliente)
  {
    try
    {
      var stream = cliente.GetStream();
      byte[] buffer = new byte[1024];
      int n;
      while ((n = stream.Read(buffer, 0, buffer.Length)) > 0)
      {
        string msg = Encoding.UTF8.GetString(buffer, 0, n);
        // Atualiza log na interface
        richTextBoxLog.Invoke((MethodInvoker)(() =>
        {
          richTextBoxLog.AppendText("Recebido: " + msg + " ");
        }));
        // Retransmite para todos
        lock (clientes)
        {
          foreach (var cli in clientes)
          {
            if (cli != cliente)
            {
              try { cli.GetStream().Write(buffer, 0, n); } catch { }
            }
          }
        }
      }
    }
    catch { }
    lock (clientes) clientes.Remove(cliente);
    cliente.Close();
  }
}
```

4. Cliente WinForms

O cliente deve conter:

- Campos para IP e porta do servidor;
- Botão para conectar;

- Caixa de texto para digitar mensagem;
- Botão para enviar mensagem;
- Área (RichTextBox) para exibir mensagens do chat.

A thread de recebimento de mensagens do socket utiliza Invoke para atualizar a interface.

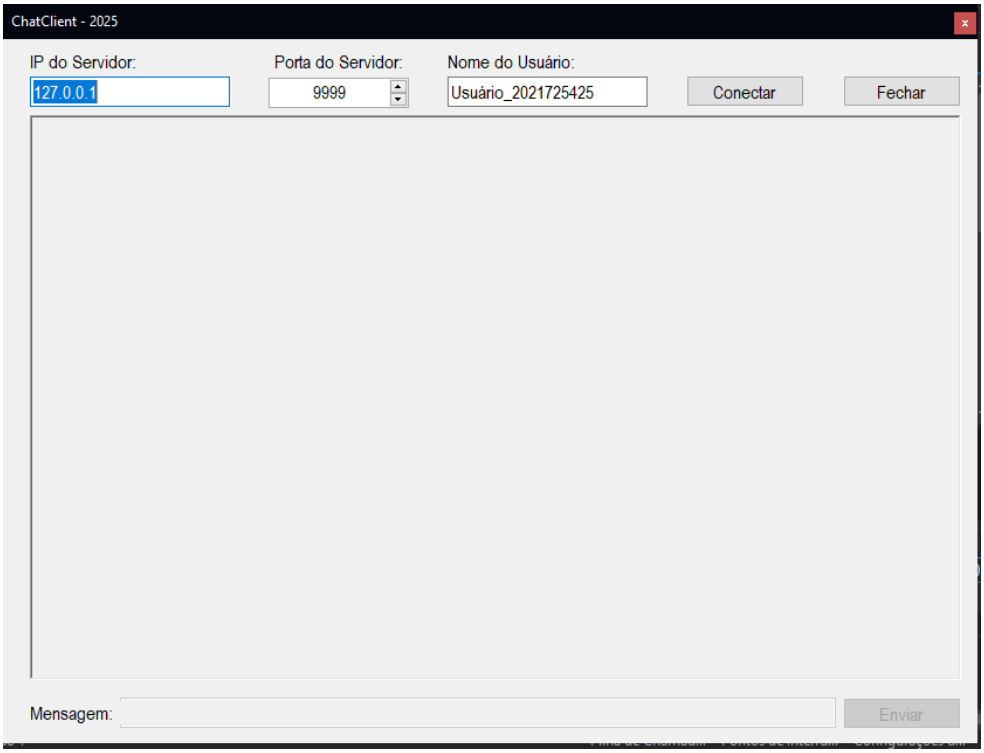
Exemplo de código para o formulário principal do cliente:

```
using System;
using System.Net.Sockets;
using System.Text;
using System.Threading;
using System.Windows.Forms;

public partial class ClienteForm : Form
{
  TcpClient cliente;
  NetworkStream stream;
  Thread threadReceber;

  public ClienteForm()
  {
    InitializeComponent();
  }

  private void buttonConectar_Click(object sender, EventArgs e)
  {
    string ip = textBoxIP.Text;
    int porta = int.Parse(textBoxPorta.Text);
    cliente = new TcpClient();
    cliente.Connect(ip, porta);
    stream = cliente.GetStream();
    threadReceber = new Thread(ReceberMensagens);
    threadReceber.IsBackground = true;
    threadReceber.Start();
    richTextBoxChat.AppendText("Conectado!");
  }

  private void buttonEnviar_Click(object sender, EventArgs e)
  {
    if (stream == null) return;
    string mensagem = textBoxMensagem.Text;
    if (string.IsNullOrWhiteSpace(mensagem)) return;
```

```
        byte[] dados = Encoding.UTF8.GetBytes(mensagem);
        stream.Write(dados, 0, dados.Length);
        textBoxMensagem.Clear();
    }

    void ReceberMensagens()
    {
        try
        {
            byte[] buffer = new byte[1024];
            int n;
            while ((n = stream.Read(buffer, 0, buffer.Length)) > 0)
            {
                string msg = Encoding.UTF8.GetString(buffer, 0, n);
                richTextBoxChat.Invoke((MethodInvoker)(() =>
                {
                    richTextBoxChat.AppendText("Recebido: " + msg + " ");
                }));
            }
        }
        catch (Exception ex)
        {
            this.Invoke((MethodInvoker)(() =>
            {
                richTextBoxChat.AppendText("Desconectado.");
            }));
        }
    }
}
```

5. Esboço da Interface do Cliente

Abaixo está um exemplo de esboço visual para a interface do cliente do chat. Inclua um RichTextBox para mensagens, TextBoxes para IP, porta e mensagem, e botões para conectar e enviar.



Sugestão de layout: adapte conforme a sua preferência.

6. Resumo e Dicas

- Use sempre 'Invoke' para atualizar a interface a partir de threads!
- Separe o envio e o recebimento de mensagens para não travar a interface.
- Lembre-se de tratar erros e desconexões.
- Teste liberando a porta no firewall para conexões externas.
- Adapte o visual conforme o objetivo da atividade ou experimento.