Sistema de Controle de Produção e de Consumo de Energia Elétrica em Campus Inteligente com Comunicação Simultânea de Usuários

Execução: Individual
Data de entrega: 29 de Maio de 2024 até 23h:59min

Versão 2.0 - Última atualização em 06/05/2024

INTRODUÇÃO

O surgimento de campi inteligentes contribui ao monitoramento de diversas informações no contexto do ambiente universitário, como a qualidade do ar, os parâmetros ambientais, a eficiência energética, a produção de energia sustentável e o consumo de energia de elétrica nas diversas instalações a fim de aprimorar a experiência dos seus frequentadores, composto por professores, técnicos e estudantes universitários, entre outros. Portanto, um campus inteligente oferece uma experiência interativa benéfica ao utilizar infraestrutura de comunicações modernas e dispositivos conectados para melhorar a cooperação, eficiência de recursos e segurança da instituição. Os administradores da instituição podem se aproveitar de campus inteligente para integrar sistemas de iluminação, geração de energia elétrica sustentável e controle de consumo elétrico para prover experiências conectadas aos estudantes.

Uma empresa de controle e automação de *Smart Campus,* soube do seu trabalho no desenvolvimento do sistema de operação de salas de aula e resolveu lhe contratar para desenvolver um projeto piloto que provê um ambiente de controle de informações elétricas de uma universidade. Basicamente, este ambiente deve consultar **o servidor** na Subestação de Energia (SE), que contém o estado das estações de produção de energia elétrica responsáveis por todo o monitoramento do sistema de geração de energia solar fotovoltaica. Bem como consultar **o servidor** do Sistema de Controle de Iluminação Inteligente (SCII), que tem o objetivo de controlar os postes de iluminação inteligentes que proveem iluminação e informações em tempo-real de consumo elétrico no campus. Além disso, o ambiente deve dispor de Interfaces de Controle (IC) com acesso à SE e ao SCII **(os clientes),** cuja função é consultar os dados do estado de produção de energia elétrica e os dados do sistema de iluminação inteligente no campus inteligente por meio de protocolos de comunicação. Esta comunicação ocorre através da Internet. A Figura 1 ilustra a comunicação entre cada uma das entidades do projeto (SE, SCII e IC).

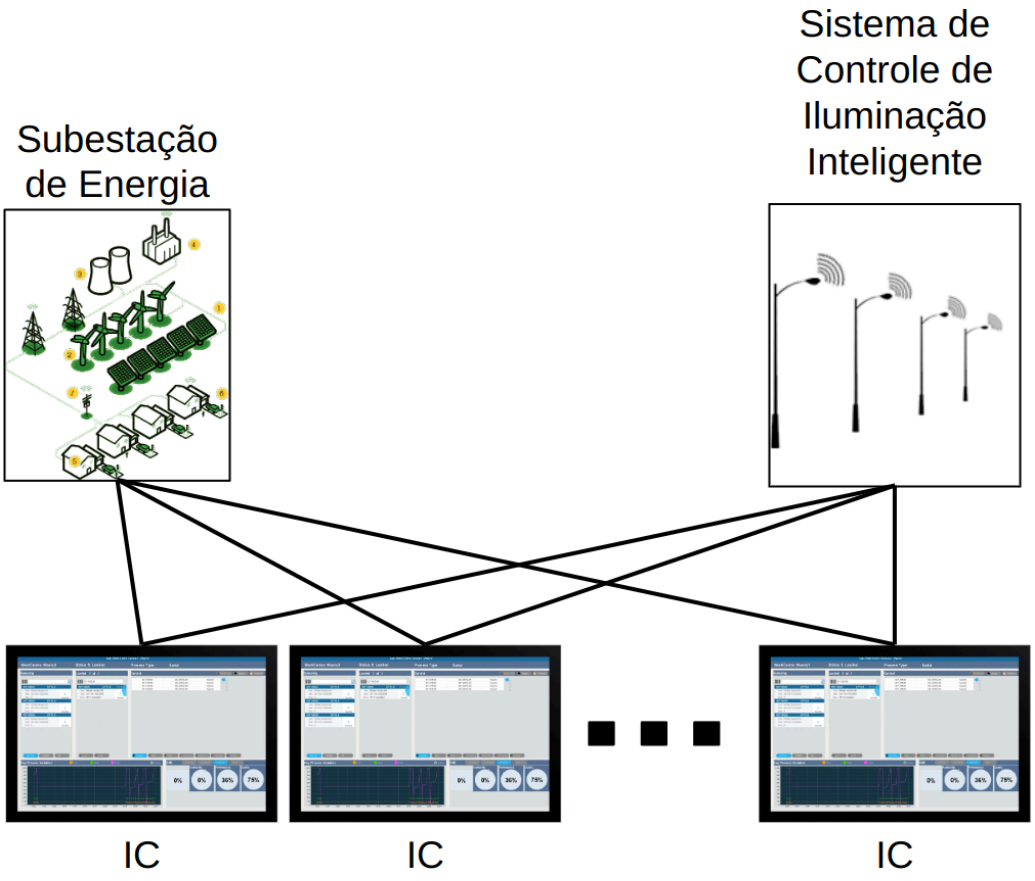

Figura 1 - Exemplo de comunicação entre as entidades do projeto

Nesse contexto, você foi contratado como Engenheiro de Automação/Sistemas para atuar no projeto em questão. A sua tarefa é desenvolver um sistema de comunicação entre os clientes IC e os servidores SE e SCII que permite o **monitoramento e controle em tempo-real** do sistema de controle de iluminação inteligente. Esse sistema deve permitir a troca de informações e de comandos entre os diferentes equipamentos e processos envolvidos, facilitando a coordenação dos recursos e otimizando o consumo elétrico.

O programa **servidor SE** deve armazenar dados **aleatórios** de produção de energia elétrica de, pelo menos, 20 (vinte) mWh e, no máximo, 50 (cinquenta) mWh, enquanto o programa **servidor SCII** deve guardar o consumo elétrico do campus inteligente que varia entre 0% e 100%, de acordo com o exemplo na **Tabela I**.

Tabela I - Dados armazenados de servidores

Campus Inteligente		
Exemplos	**Produção de Energia Elétrica = [20, 50]* mWh**	**Consumo Elétrico = [0,100]* %**
1	27 mWh	30%
2	21 mWh	100%
3	49 mWh	67%

* Os valores aleatórios devem ser números inteiros que respeitem o intervalo estipulado para cada parâmetro.

Em outras palavras, você deve implementar um **sistema** em redes caracterizado pela existência de dois **servidores (SE e SCII).** Além disso, cada um desses servidores é responsável por estabelecer uma **conexão passiva** com os **clientes (ICs)**, pelo **gerenciamento de múltiplas conexões** com os seus clientes, pela **intermediação das mensagens** enviadas pelos clientes, pelo **encerramento passivo de conexão** com seus clientes e pelo seu próprio **encerramento de conexão.**

Os **equipamentos ICs** desempenham o papel de **clientes,** sendo responsáveis pelo **estabelecimento ativo de conexão**, necessariamente, com ambos os servidores, pelo **envio e recebimento de mensagens** trocadas com os servidores e pelo **encerramento ativo de conexão.**

Toda conexão deve utilizar a interface de *sockets* na linguagem C.

Você desenvolverá os dois (2) programas para um sistema simples de troca de mensagens empregando apenas as funcionalidades da biblioteca de *sockets* POSIX e a comunicação via protocolo. O programa referente ao servidor deverá usar **um** socket para a conexão passiva com os clientes. Deve-se utilizar a função **select()** da biblioteca de sockets para o gerenciamento das múltiplas conexões estabelecidas. As próximas seções detalham o que cada entidade (servidor e cliente) deve fazer.

Os objetivos gerais deste trabalho são:

1. Implementar um servidor usando a interface de *sockets* na linguagem C*;*
2. Implementar um cliente usando a interface de *sockets* na linguagem C*;*
3. Escrever o relatório.

PROTOCOLO

O protocolo de aplicação deve funcionar sobre o protocolo TCP. Isso implica que as mensagens são entregues sobre um canal de bytes com garantias de entrega em ordem, mas é sua responsabilidade

implementar as especificações das mensagens e as funcionalidades tanto dos servidores quanto dos clientes.

Os servidores e os clientes trocam mensagens curtas de até 500 bytes usando o transporte TCP. As mensagens carregam textos codificados segundo a tabela ASCII. Apenas letras, números e espaços podem ser transmitidos. Caracteres acentuados e especiais não devem ser transmitidos.

Especificações das Mensagens

Esta seção especifica as mensagens padrões na comunicação de controle e dados da rede, bem como as mensagens de erro e confirmação. Nas tabelas abaixo, as células em "–" correspondem aos campos que não precisam ser definidos nas mensagens.

Mensagens de Controle		
Tipo	**Payload**	**Descrição**
REQ_ADD	–	Mensagem de requisição de entrada de cliente na rede
RES_ADD	IdC_i	Mensagem de resposta de identificação IdC_i do cliente C_i
REQ_REM	IdC_i	Mensagem de requisição de saída de cliente na rede, onde IdC_i corresponde à identificação do cliente solicitante

Mensagens de Dados		
Tipo	**Payload**	**Descrição**
REQ_INFOSE	–	Mensagem de requisição de informações de produção de energia elétrica
RES_INFOSE	**value**	Mensagem de resposta de informações de produção de energia elétrica
REQ_INFOSCII	–	Mensagem de requisição de informações de consumo elétrico
RES_INFOSCII	**value**	Mensagem de resposta de informações de consumo elétrico
REQ_STATUS	–	Mensagem de requisição de estado da produção de energia elétrica
RES_STATUS	**status**	Mensagem de resposta de estado da produção de energia elétrica
REQ_UP	–	Mensagem de requisição de aumento no consumo elétrico

RES_UP	**old_value current_value**	Mensagem de resposta de aumento no consumo elétrico
REQ_NONE	–	Mensagem de requisição de inalteração no consumo elétrico
RES_NONE	**current_value**	Mensagem de resposta de inalteração no consumo elétrico
REQ_DOWN	–	Mensagem de requisição de redução no consumo elétrico
RES_DOWN	**old_value current_value**	Mensagem de resposta de redução no consumo elétrico

Mensagens de Erro ou Confirmação		
Tipo	**Payload**	**Descrição**
ERROR	Code	Mensagem de erro transmitida do Servidor para cliente C_i. O campo payload informa o código de erro. Abaixo descrição de cada código: 01: "*Client limit exceeded*" 02: *"Client not found"*
OK	Code	Mensagem de confirmação transmitida do Servidor para cliente C_i. O campo payload informa o código de confirmação. Abaixo descrição de cada código: 01: "Successful disconnect"

Fluxo das Mensagens de Controle

Esta seção descreve o fluxo de mensagens de controle transmitidas entre cliente-servidores a fim de coordenar a comunicação dos clientes na rede. Além das decisões e impressões em tela realizadas pelos servidores e clientes.

Abertura de comunicação de Cliente com Servidores

1. Um cliente **C_i** solicita aos servidores **SE** e **SCII** a abertura de comunicação a fim de obter seu identificador na rede.
2. Os servidores **SE** e **SCII** recebem requisição de **C_i** e verifica se quantidade máxima de conexões foi alcançada.
    2.1. Em caso positivo, os servidores **SE** e **SCII** imprimem a mensagem de erro **ERROR(01)** (vide *Especificação das Mensagens*).
        2.1.1. Cliente **C_i** imprime em tela descrição do código de erro **ERROR(01)**.
    2.2. Em caso negativo, cliente C_i envia a mensagem **REQ_ADD** para os servidores **SE** e **SCII.**
        2.2.1. Os servidores **SE** e **SCII** definem um identificador **IdC_i** para **C_i** único entre os seus clientes, registra o identificador em sua base de dados, imprime em tela a mensagem "*Client IdC_i added*" e envia para o cliente C_i a mensagem **RES_ADD(IdC_i).**

O cliente **C_i,** ao receber as mensagens **RES_ADD(IdC_i)** dos servidores **SE** e **SCII**, registra sua nova identificação e imprime em tela ambas as mensagens

Servidor SE New ID: IdC_i
Servidor SCII New ID: IdC_i

Fechamento de comunicação de Cliente com Servidores

1. Um cliente **C_i** (IC) recebe comando via teclado

   kill

   e solicita aos servidores **SE** e **SCII** o fechamento da comunicação por meio da mensagem **REQ_REM(IdC_i)**.
2. Os servidores **SE** e **SCII** recebem **REQ_REM(IdC_i)** e verificam se **IdC_i** existe na base de dados.
   2.1. Em caso negativo, os servidores **SE** e **SCII** respondem mensagem de erro **ERROR(02)** para cliente **C_i**.
      2.1.1. Cliente **C_i** recebe mensagem **ERROR(02)** dos servidores **SE** e **SCII** e imprime em tela a descrição do código de erro correspondente (vide *Especificação das Mensagens*).
   2.2. Em caso positivo, os servidores **SE** e **SCII** removem **C_i** da base de dados, respondem mensagem **OK(01)** para **C_i,** desconecta **C_i,** imprime em tela as mensagens, respectivamente,

   Servidor SE Client IdC_i removed
   Servidor SCII Client IdC_i removed
      2.2.1. Cliente **C_i** recebe mensagens **OK(01)** de confirmação e imprime em tela a descrição da mensagem, fecha a conexão e encerra a execução.

Descrição das Funcionalidades

Esta seção descreve o fluxo de mensagens transmitidas entre os clientes (ICs) e os servidores (SE e SCII) resultante de cada uma das três funcionalidades da aplicação a fim de monitorar e controlar as informações de produção e consumo de energia elétrica.

1) Consultar informações de produção de energia elétrica

1. Um cliente **C_i** (IC) recebe comando via teclado

   display info se

   para consultar os valores de produção de energia elétrica. Para isso, o cliente **C_i** envia a mensagem **REQ_INFOSE** para o servidor **SE**.
2. O servidor **SE** recebe a solicitação e consulta as informações de produção de energia elétrica em sua base de dados.
   2.1. O servidor **SE** responde o cliente **C_i** com o valor de produção de energia elétrica por meio da mensagem **RES_INFOSE**.
      2.1.1. O cliente **C_i** recebe mensagem e imprime em tela:

      producao atual: **value**

2) Consultar informações de consumo elétrico

1. Um cliente **C_i** recebe comando via teclado

   display info scii

   para consultar os valores de consumo elétrico. Para isso, o cliente **C_i** envia a mensagem **REQ_INFOSCII** para o servidor **SCII**.

2. O servidor **SCII** recebe a solicitação e consulta as informações de consumo elétrico em sua base de dados.
    2.1. O servidor **SCII** responde o cliente $\mathbf{C_i}$ com o valor de consumo elétrico por meio da mensagem **RES_INFOSCII**.
        2.1.1. O cliente $\mathbf{C_i}$ recebe mensagem e imprime em tela:
            consumo atual: **value**

3) Manutenção da produção de energia elétrica e consumo elétrico

1. Um cliente $\mathbf{C_i}$ recebe comando via teclado
    query condition
    para consultar o estado da produção de energia elétrica. Para isso, o cliente $\mathbf{C_i}$ envia a mensagem **REQ_STATUS** para o servidor **SE**.
2. O servidor **SE** recebe a solicitação, consulta as informações da produção de energia elétrica em sua base de dados e verifica o estado da produção atual de energia elétrica:
    2.1. Possíveis estados:
        2.1.1. Maior ou igual a 41 kWh, o estado é **alta**
        2.1.2. Entre 31 kWh e 40 kWh, o estado é **moderada**
        2.1.3. Entre 20 kWh e 30 kWh, o estado é **baixa**

    2.2. O servidor **SE** responde o cliente $\mathbf{C_i}$ com o valor do estado atual da produção de energia elétrica por meio da mensagem **RES_STATUS**.
        2.2.1. O cliente $\mathbf{C_i}$ recebe mensagem e imprime em tela:
            estado atual: **status**
        2.2.2. Além disso, o cliente $\mathbf{C_i}$ verifica qual o estado atual da produção de energia elétrica:
            2.2.2.1. Se for **alta**, o cliente $\mathbf{C_i}$ envia a mensagem **REQ_UP** para o servidor **SCII** a fim de informar a possibilidade de aumento do consumo elétrico**.**
                2.2.2.1.1. O servidor **SCII** recebe a solicitação, consulta as informações de consumo elétrico em sua base de dados, guarda essa informação **(old_value)**, busca aumentar aleatoriamente o valor do consumo elétrico **new_value** e atualiza sua base de dados com o novo valor para o consumo elétrico, respeitando os valores estabelecidos na **Tabela I**. Note que **new_value** deve ser obrigatoriamente maior ou igual a **old_value.**
                2.2.2.1.2. O servidor **SCII** responde o cliente $\mathbf{C_i}$ com os valores antigo e atual de consumo elétrico por meio da mensagem **RES_UP**.
                    2.2.2.1.2.1. O cliente $\mathbf{C_i}$ recebe mensagem e imprime em tela:
                        consumo antigo: **old_value**
                        consumo atual: **new_value**
            2.2.2.2. Se for **moderada**, o cliente $\mathbf{C_i}$ envia a mensagem **REQ_NONE** para o servidor **SCII** a fim de informar que não deve haver alteração no consumo elétrico.
                2.2.2.2.1. O servidor **SCII** recebe a solicitação, consulta as informações de consumo elétrico em sua base de dados **(current_value)** e responde o cliente $\mathbf{C_i}$ com o valor atual de consumo elétrico por meio da mensagem **RES_NONE**.
                    2.2.2.2.1.1. O cliente $\mathbf{C_i}$ recebe mensagem e imprime em tela:
                        consumo antigo: **current_value**
            2.2.2.3. Se for **baixa**, o cliente $\mathbf{C_i}$ envia a mensagem **REQ_DOWN** para o servidor **SCII** a fim de informar a possibilidade de redução do consumo elétrico**.**

2.2.2.3.1. O servidor **SCII** recebe a solicitação, consulta as informações de consumo elétrico em sua base de dados, guarda essa informação **(old_value)**, busca diminuir aleatoriamente o valor do consumo elétrico **new_value** e atualiza sua base de dados com o novo valor para o consumo elétrico, respeitando os valores estabelecidos na **Tabela I**. Note que **new_value** deve ser obrigatoriamente menor ou igual a **old_value.**

2.2.2.3.2. O servidor **SCII** responde o cliente $\mathbf{C_i}$ com os valores antigo e atual de consumo elétrico por meio da mensagem **RES_DOWN**.

2.2.2.3.2.1. O cliente $\mathbf{C_i}$ recebe mensagem e imprime em tela:

consumo antigo: **old_value**
consumo atual: **new_value**

2.3. O servidor **SE** deve gerar um novo valor aleatório para a produção de energia elétrica, respeitando os valores estabelecidos na **Tabela I**, e armazenar em sua base de dados.

IMPLEMENTAÇÃO

Os seguintes detalhes devem ser observados no desenvolvimento de cada programa que fará parte do sistema. É importante observar que o protocolo é simples e único (o cliente sempre tem que enviar a mensagem para os servidores e vice-versa, de modo que o correto entendimento da mensagem deve ser feito por todos os programas).

Como mencionado anteriormente, a implementação do protocolo da aplicação utilizará a comunicação TCP. Haverá **dois** sockets em cada cliente, um para se comunicar com o **servidor SE** e outro para se comunicar com o **servidor SCII**, independente de quantos outros programas se comunicarem com aquele processo. Já os servidores inicializam-se com **um** socket cada. À medida que se conecta e se desconecta de clientes, outros sockets são adicionados/descartados do seu pool de sockets.

O tipo de endereço IP assumido neste trabalho prático deve ser **IPv4 e IPV6**. Um número máximo de **2 servidores** serão executados simultaneamente. Cada **servidor** deve tratar até **10 clientes simultaneamente**. Ademais, os servidores são responsáveis por definir identificações únicas para cada cliente na rede. Um **cliente** inicia sem identificação, após a solicitação de entrada na rede ele recebe sua identificação. Além disso, o cliente e os servidores devem receber mensagens do teclado.

Outros detalhes de implementação:

- Os servidores devem encerrar apenas a conexão com o cliente ao receber a mensagem **`“kill”`** a qualquer momento
- Cada mensagem possui no máximo 500 bytes

Execução

Seus servidores devem receber um número de porta na linha de comando especificando em qual porta ele vai estabelecer as conexões dos clientes. Para padronização do trabalho, utilize a porta **12345** para a conexão com o **servidor SE** e a porta **54321** para se conectar com o **servidor SCII**. Seu cliente deve receber, **estritamente nessa ordem**, o endereço IP e as portas dos servidores **SE e SCII**, respectivamente, em que ele deseja se conectar para o estabelecimento da comunicação. Para realizar múltiplas conexões de clientes com o servidor basta executar múltiplas vezes o código do programa cliente.

A seguir, um exemplo de execução de dois clientes conectados com dois servidores em quatro terminais distintos, considerando ambos os protocolos IPv4 e IPv6, respectivamente:

```
Terminal 1: ./server v4 12345
Terminal 2: ./server v4 54321
Terminal 3: ./client 127.0.0.1 12345 54321
Terminal 4: ./client 127.0.0.1 12345 54321
```

ou

```
Terminal 1: ./server v6 12345
Terminal 2: ./server v6 54321
Terminal 3: ./client ::1 12345 54321
Terminal 4: ./client ::1 12345 54321
```

AVALIAÇÃO

O trabalho deve ser realizado individualmente e **deve ser implementado na linguagem de programação C** utilizando somente a biblioteca padrão (interface POSIX de sockets de redes). Deve ser possível executar seu programa no sistema operacional **Linux** e **não deve utilizar bibliotecas Windows, como o winsock**. Procure escrever seu código de maneira clara, com comentários pontuais e bem indentados. Isto facilita a correção dos monitores e tem impacto positivo na avaliação.

Correção Semi-automática

Seu servidor será corrigido de forma semi-automática por uma bateria de testes. Cada teste verifica uma funcionalidade específica do servidor. Os testes avaliam a aderência do seu servidor ao protocolo de comunicação inteiramente através dos dados trocados através da rede (a saída dos seus programas na tela, e.g., para depuração, não impacta os resultados dos testes).

Para a correção os seguintes testes serão realizados **(com IPv4 e IPv6)**:

- Abertura de comunicação cliente-servidores: **+2 pontos**
- Fechamento de comunicação cliente-servidores: **+2 pontos**
- Consultar informações de produção de energia elétrica: **+5 pontos**
- Consultar informações de consumo elétrico: **+5 pontos**
- Manutenção da produção de energia elétrica e consumo elétrico: **+8 pontos**
- Cliente envia kill para o servidor e encerra a execução do programa cliente: **+2 ponto**

Total: **24 pontos (80%)** + 6 pontos (20%) (documentação)

Os testes serão executados com ambos os protocolos, IPv4 e IPv6. Caso um destes não esteja implementado, a nota sofrerá penalização de 50%, ou seja, até 12 pontos, visto que serão executados apenas metade dos possíveis testes.

Entrega

Cada aluno deve entregar documentação em PDF de até 6 páginas, sem capa, utilizando fonte tamanho 10, e figuras de tamanho adequado ao tamanho da fonte. **Ele deve conter uma descrição da arquitetura**

adotada para o servidor, os refinamentos das ações identificadas no mesmo, as estruturas de dados utilizadas, as decisões de implementação não documentadas nesta especificação. Como sugestão, considere incluir as seguintes seções no relatório: introdução, mensagens, arquitetura, servidor, cliente, discussão e conclusão. O relatório deve ser entregue em formato PDF. A documentação corresponde a 20% dos pontos do trabalho (**+6 pontos**), mas só será considerada para as funcionalidades implementadas corretamente.

Será utilizado um sistema para detecção de código repetido, portanto não é admitido cola de trabalhos. Se uma das partes do trabalho não for entregue (código ou relatório) a nota final será zero.

Cada aluno deve entregar, além da documentação, o **código fonte em C** e um **Makefile** para compilação do programa. Instruções para submissão:

- O Makefile deve compilar o "client" e o "server".
- Seu código deve ser compilado pelo comando "make" sem a necessidade de parâmetros adicionais.
- A entrega deve ser feita no formato ZIP, seguindo a nomenclatura: TP_MATRICULA.zip
- O nome dos arquivos deve ser padronizado:
    - server.c
    - client.c
    - common.c, common.h (se houver)

Prazo de entrega

Os trabalhos poderão ser entregues até às 23:59 (vinte e três e cinquenta e nove) do dia especificado para a entrega. O horário de entrega deve respeitar o relógio do sistema Moodle, ou seja, a partir de 00:00 do dia seguinte à entrega no relógio do Moodle, os trabalhos **não poderão ser entregues**. Logo não serão considerados trabalhos entregues fora do prazo definido.

Pedidos de Revisão de Notas

Os alunos poderão realizar pedidos de revisão de nota em até três dias corridos contados a partir da data da liberação da nota. Em caso de solicitação de revisão, o aluno deverá enviar o pedido para o e-mail do professor com as seguintes orientações:

- No título do email o rótulo [TP2-Rev] - nome e número de matrícula.
- No corpo da mensagem a sua questão propriamente dita.

Dicas e Cuidados

- O guia de programação em rede do Beej (http://beej.us/guide/bgnet/) tem bons exemplos de como organizar um servidor
- Procure escrever seu código de maneira clara, com comentários pontuais e bem identado.
- Não se esqueça de conferir se seu código não possui erros de compilação ou de execução.
- Implemente o trabalho por partes. Por exemplo, implemente o tratamento das múltiplas conexões, depois crie os formatos das mensagens e, por fim, trate as mensagens no servidor ou no cliente.

EXEMPLOS DE EXECUÇÃO

Exemplos de execução serão disponibilizados brevemente.

LEMBRETES

- Para usuários Windows, a ferramenta Windows Subsystems for Linux (WSL), nativa do sistema operacional, possibilita operar ambientes Linux no Windows. Seguem os links sobre a ferramenta e tutorial para instalação:
  - https://learn.microsoft.com/pt-br/windows/wsl/about
  - https://learn.microsoft.com/pt-br/windows/wsl/install
- Aos alunos que preferem operar diretamente em ambientes Linux, o ICEx possui duas salas (1006 e 1008) com computadores com sistema operacional Linux disponíveis a todos estudantes da UFMG. Essas salas ficam localizadas no térreo, à direita da portaria principal do ICEx.