Sistema de Controle de Acesso aos Prédios Universitários (v1.2)

Execução: Individual
Data de entrega: Até 23h59min da data publicada no Moodle

ATENÇÃO: Este documento sofreu alterações desde a sua última versão. Essas alterações estão coloridas na cor desta frase e marcadas pelo caractere ※, também nesta cor.

INTRODUÇÃO

Buscando aprimorar a segurança em seus campi, uma universidade decidiu adotar um sistema de controle de acesso por meio de catracas inteligentes. A tecnologia permitirá não apenas restringir a entrada e saída de pessoas, mas também rastrear a localização dos indivíduos em cada prédio baseado nos registros das catracas. Para implementar um projeto piloto, a instituição contratou uma empresa especializada em controle de acesso.

A proposta da empresa considera que em cada prédio da universidade seja instalado um computador, denominado Interface de Controle (IC), responsável por se comunicar com as catracas de suas portarias para autorizar o trânsito de pessoas e registrar a localização dos indivíduos. Além disso, os funcionários da portaria usarão o sistema para cadastrar as pessoas autorizadas a entrar nos prédios e para localizá-las dentro do campus. Para fazer isso, este computador se comunicará com **dois servidores: o servidor de usuários e o servidor de localização.**

O servidor de usuários deve guardar a lista de pessoas cadastradas e autenticar essas pessoas quando solicitado. Já o servidor de localização deve guardar o registro de localização das pessoas que acessaram os prédios do campus. Por simplicidade, neste projeto piloto não será necessário tratar a exclusão de pessoas de nenhuma dessas bases de dados. Para tornar o protocolo mais robusto, estes servidores se comunicarão entre si para cumprir algumas de suas funções.

Como as catracas inteligentes são dispositivos caros e de entrega demorada, a universidade optou por fazer o projeto piloto com catracas simuladas, na esperança de que futuramente a integração seja simples. Dessa forma, o sistema a ser desenvolvido contará com **2 servidores**: o servidor de usuários (SU) e o servidor de localização (SL), e **um software de IC (cliente)**, que receberá todos os comandos, incluindo os comandos de entrada e saída de pessoas, pela entrada padrão.

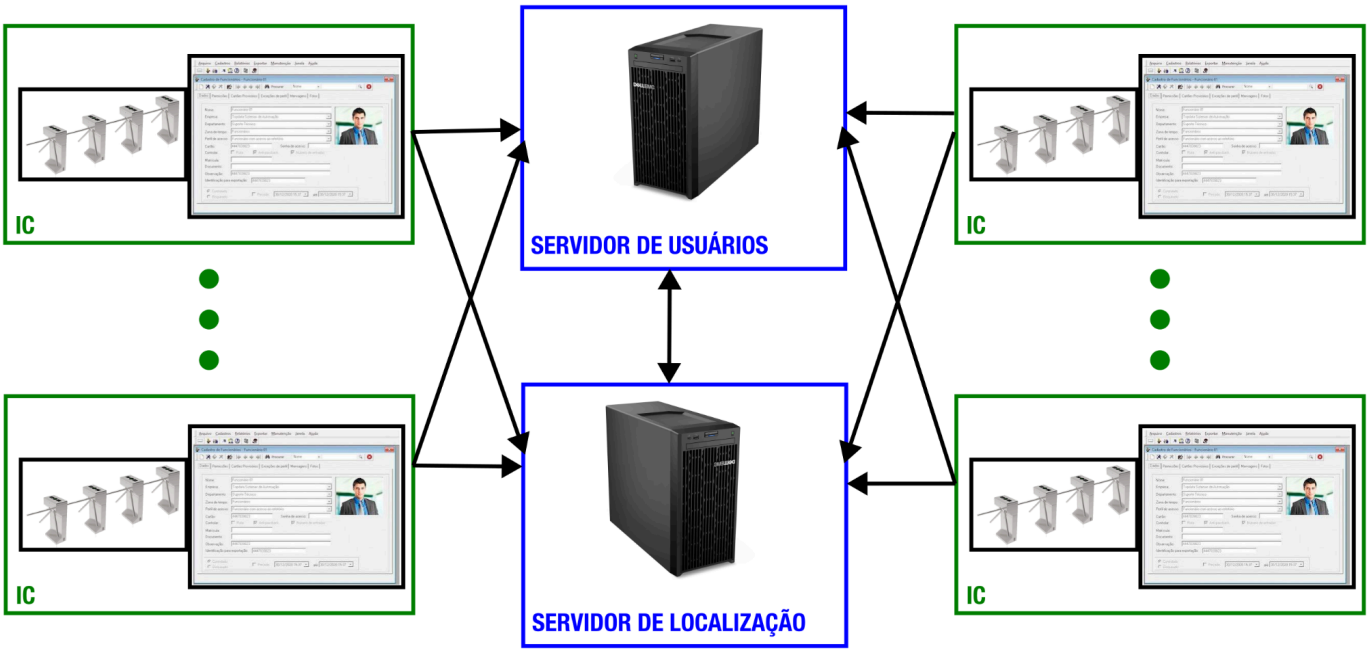

Figura 1 - Exemplo de comunicação entre as entidades do sistema. A direção da seta indica o modo de conexão: a origem se conecta ativamente ao destino. Entre os servidores há uma seta dupla para indicar que a conexão é Peer-to-Peer.

Nesse contexto, em vista do seu excelente currículo na área de Redes de Computadores, você foi contratado como Engenheiro de Sistemas pela empresa responsável para atuar no projeto em questão. A sua tarefa é desenvolver um sistema de comunicação entre os clientes IC e os servidores SU e SL que permita o **monitoramento e controle de acesso em tempo-real** das pessoas dentro do campus da

universidade. Esse sistema deve permitir a troca de informações e de comandos entre os diferentes equipamentos e processos envolvidos.

O programa **servidor de usuários (SU)** deve armazenar os IDs dos usuários cadastrados, e incluir uma flag informando se o usuário tem permissão especial, conforme o exemplo na **Tabela I**. Já o programa **servidor de localização (SL)** deve guardar a última localização das pessoas, conforme o exemplo na **Tabela II**.

Tabela I - Dados armazenados no servidor de usuários

Exemplos	ID do usuário (10 caracteres numéricos)	Tem permissão especial (0 ou 1)
1	2042010377	0
2	2042980421	1
3	2041567983	0

Tabela II - Dados armazenados no servidor de localização

Exemplos	ID do usuário (10 caracteres numéricos)	Última localização (int) ※
1	2042010377	1
2	2042980421	2
3	2041567983	-1

※ O valor da localização deve ser um inteiro **entre 1 e 10** para indicando os prédios vinculados às ICs **ou -1** para indicar a localização “fora de qualquer prédio”.

Em outras palavras, você deve implementar um **sistema** caracterizado pela existência de dois **servidores** (SU e SL) conectados entre si através de uma abordagem **peer-2-peer,** i.e. onde ambos os servidores podem realizar o papel **passivo ou ativo** ao se conectarem. Além disso, cada um desses servidores será responsável por estabelecer uma **conexão passiva** com as **ICs** de cada prédio da universidade, pelo **gerenciamento de múltiplas conexões** com os seus clientes, pelo **encerramento passivo de conexão** com seus clientes e pelo **encerramento de conexão** com o seu peer**.**

As **ICs** desempenharão o papel de **clientes,** sendo responsáveis pelo **estabelecimento ativo de conexão** com os dois servidores, pelo **envio e recebimento de mensagens** trocadas com estes 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 utilizando apenas as funcionalidades da biblioteca de *sockets* POSIX e a comunicação via protocolo. O programa referente ao servidor deverá usar **dois** sockets, um para a conexão com o outro servidor e outro 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 ou 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 a documentação.

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 e (Direção)	**Código**	**Payload**	**Descrição**
REQ_CONNPEER (S_i -> S_j)	17	–	Mensagem de requisição de conexão entre peers
RES_CONNPEER (S_j -> S_i)	18	$PidS_i$	Mensagem de resposta de conexão entre peers
REQ_DISCPEER (S_i -> S_j)	19	$PidS_i$	Mensagem de requisição de encerramento de conexão entre peers
REQ_CONN (C_i -> S_i)	20	LocId	Mensagem de requisição de entrada de cliente na rede
RES_CONN (S_i -> C_i)	21	IdC_i	Mensagem de resposta de entrada de cliente na rede, com a identificação IdC_i do cliente C_i
REQ_DISC (C_i -> S_i)	22	IdC_i	Mensagem de requisição de saída de cliente na rede, onde IdC_i corresponde à identificação do cliente solicitante

Mensagens de Dados			
Tipo e (Direção)	**Código**	**Payload**	**Descrição**
REQ_USRADD (C_i -> SU)	33	UID, is_special	Mensagem de requisição de cadastro de pessoa

REQ_USRACCESS (C_i -> SU)	34	UID, direction	Mensagem de requisição de entrada e saída de pessoas (direction = *in* / *out*)
RES_USRACCESS (SU -> C_i)	35	LocId	Mensagem de resposta da entrada e saída de pessoas, contendo o id da última localização
REQ_LOCREG (SU -> SL)	36	UID, LocId	Mensagem de requisição de registro da localização de uma pessoa
RES_LOCREG (SL -> SU)	37	LocId	Mensagem de resposta de registro da localização de uma pessoa, contendo o id da última localização
REQ_USRLOC (C_i -> SL)	38	UID	Mensagem de requisição de informação sobre a localização de uma pessoa
RES_USRLOC (SL -> C_i)	39	LocId	Mensagem de resposta de informação sobre a localização de uma pessoa
REQ_LOCLIST (C_i -> SL)	40	UID, LocId	Mensagem de requisição de informação sobre as pessoas presentes em uma localização (deve autenticar usuário com o UID)
RES_LOCLIST (SL -> C_i)	41	UIDs	Mensagem de resposta de informação sobre as pessoas presentes em uma localização
REQ_USRAUTH (SL -> SU)	42	UID	Mensagem de requisição de autenticação de pessoa com UID
RES_USRAUTH (SU -> SL)	43	is_special	Mensagem de resposta de localização de pessoas

Mensagens de Erro ou Confirmação			
Tipo	**Código**	**Payload**	**Descrição**
ERROR	255	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: "*Peer limit exceeded*" 02: "*Peer not found*" 09: "*Client limit exceeded*" 10: "*Client not found*" 17: "*User limit exceeded*" 18: "*User not found*" 19: "*Permission denied*"

OK	0	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*" 02: "*Successful create*" 03: "*Successful update*"

Fluxo das mensagens de controle

Esta seção descreve o fluxo de mensagens de controle transmitidas entre servidor-servidor e entre cliente-servidor 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 entre Servidores (Peer-2-Peer)

1. O **servidor S_i** tenta solicitar ao **servidor S_j** a abertura de comunicação por meio da mensagem **REQ_CONNPEER()**
    1.1. Caso não haja um servidor **S_j** aberto para conexão, o servidor **S_i** imprime em tela a mensagem *"No peer found, starting to listen..."* e começa a ouvir potenciais novas conexões.
    1.2. Caso haja um servidor **S_j** aberto para conexão, ele recebe a requisição de **S_i** e verifica se a quantidade máxima de conexões peer-2-peer foi alcançada.
        1.2.1. Em caso positivo, o servidor **S_j** responde uma mensagem de **ERROR(01)** para **S_i**
            1.2.1.1. Servidor **S_i** recebe a mensagem **ERROR(01)** e imprime na tela a descrição do código de erro correspondente (vide *Especificação das Mensagens*).
        1.2.2. Em caso negativo, servidor **S_j** define um identificador **$PidS_i$** para **S_i**, registra o identificador em sua base de dados, imprime em tela a mensagem *"Peer $PidS_i$ connected"* e envia para **S_i** o identificador **$PidS_i$** definido através da mensagem **RES_CONNPEER($PidS_i$).**
            1.2.2.1. **S_i** recebe o seu identificador através da mensagem **RES_CONNPEER($PidS_i$),** imprime na tela *"New Peer ID: $PidS_i$"* define um identificador **$PidS_j$** para o servidor **S_j**, imprime em tela a mensagem *"Peer $PidS_j$ connected"* e envia para **S_j** o identificador através da mensagem **RES_CONNPEER($PidS_j$)**
            1.2.2.2. **S_j** recebe o seu identificador através da mensagem **RES_CONNPEER($PidS_j$),** imprime na tela *"New Peer ID: $PidS_j$"*.

Fechamento de comunicação entre Servidores (Peer-2-Peer)

1. O **servidor S_i** recebe comando via teclado
   kill
   e solicita ao **servidor S_j** o fechamento de comunicação através da mensagem **REQ_DISCPEER($PidS_i$)**.
2. **Servidor S_j** recebe **REQ_DISCPEER($PidS_i$)** e verifica se **$PidS_i$** é o identificador do peer conectado.
    2.1. Em caso negativo, o servidor **S_j** responde mensagem de erro **ERROR(02)** para o servidor **S_i** .
        2.1.1. Servidor **S_i** recebe mensagem de **ERROR(02)** de **S_j** e imprime na tela a descrição do código de erro correspondente (vide *Especificação das Mensagens).*
    2.2. Em caso positivo, o servidor **S_j** remove **S_i** da sua base de dados, responde mensagem **OK(01)** para **S_i** e imprime na tela a mensagem *Peer $PidS_i$ disconnected*.

2.2.1. **Servidor S_i** recebe mensagem **OK(01)** de confirmação, imprime em tela a descrição da mensagem, remove **S_j** da sua base de dados, imprime na tela a mensagem *Peer $PidS_j$ disconnected* e encerra a sua execução.

2.2.2. **Servidor S_j** imprime em tela a mensagem *"No peer found, starting to listen..."* e começa a ouvir potenciais novas conexões.

Abertura de comunicação de Cliente com Servidores

1. Um cliente **C_i** é iniciado para uma localização **LocId** e verifica se **LocId** é um valor entre **1 e 10**, inclusive.
   1.1. Caso negativo, **C_i** imprime na tela a mensagem *Invalid argument* e encerra a sua execução sem trocar nenhuma mensagem na rede.
   1.2. Caso positivo, **C_i** continua a execução de acordo com o passo 2.
2. O cliente **C_i** solicita aos servidores **SU** e **SL** a abertura de comunicação a fim de obter seu identificador na rede.
3. Os servidores **SU** e **SL** recebem requisição de **C_i** e verifica se quantidade máxima de conexões foi alcançada.
   3.1. Em caso positivo, os servidores **SU** e **SL** imprimem a mensagem de erro **ERROR(09)** (vide *Especificação das Mensagens*).
      3.1.1. Cliente **C_i** imprime em tela descrição do código de erro **ERROR(09)**.
   3.2. Em caso negativo, cliente **C_i** envia a mensagem **REQ_CONN(LocId)** para os servidores **SU** e **SL.**
      3.2.1. Os servidores **SU** e **SL** 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 (Loc **LocId**)*" e envia para o cliente **C_i** a mensagem **RES_CONN(IdC_i).** O cliente **C_i,** ao receber as mensagens **RES_CONN(IdC_i)** dos servidores **SU** e **SL,** registra sua nova identificação e imprime em tela ambas as mensagens

*SU New ID: **IdC_i***
*SL 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 **SU** e **SL** o fechamento da comunicação por meio da mensagem **REQ_DISC(IdC_i)**.
2. Os servidores **SU** e **SL** recebem **REQ_DISC(IdC_i)** e verificam se **IdC_i** existe na base de dados.
   2.1. Em caso negativo, os servidores **SU** e **SL** respondem mensagem de erro **ERROR(10)** para cliente **C_i**.
      2.1.1. Cliente **C_i** recebe mensagem **ERROR(10)** dos servidores **SU** e **SL** e imprime em tela a descrição do código de erro correspondente (vide *Especificação das Mensagens*).
   2.2. Em caso positivo, os servidores **SU** e **SL** removem **C_i** da base de dados, respondem mensagem **OK(01)** para **C_i,** desconectam **C_i,** e imprimem em tela a mensagem

*Client **IdC_i** removed (Loc **LocId**)*

      2.2.1. Cliente **C_i** recebe mensagem **OK(01)** de confirmação, imprime em tela as mensagens

SU Successful disconnect
SL Successful disconnect

para a mensagens vinda do servidor SU e SL, respectivamente, fecha as conexões e encerra a sua execução.

Descrição das funcionalidades

Esta seção descreve o fluxo de mensagens transmitidas entre os clientes (**ICs**) e os servidores (**SU** e **SL**) resultante de cada uma das quatro funcionalidades da aplicação a fim de monitorar e controlar o acesso das pessoas aos prédios do campus.

1) Cadastro de usuários no servidor de usuários (IC -> SU)

1. Um cliente $\mathbf{C_i}$ (IC) recebe comando via teclado
   add **UID IS_SPECIAL**
   para adicionar um usuário de id **UID** (exatamente 10 caracteres) que tem ou não permissão especial (**IS_SPECIAL** igual a 1 ou 0, respectivamente). Para isso, o cliente $\mathbf{C_i}$ envia a mensagem **REQ_USRADD(UID, IS_SPECIAL)** para o servidor **SU**.
2. ※ Servidor **SU** recebe a requisição de $\mathbf{C_i}$ e imprime em tela REQ_USRADD **UID IS_SPECIAL**
3. O servidor **SU** verifica se o usuário de id **UID** já existe.
   3.1. Caso positivo, o servidor **SU** atualiza o valor de **IS_SPECIAL** deste usuário e responde $\mathbf{C_i}$ com uma mensagem **OK(03)**.
      3.1.1. O cliente $\mathbf{C_i}$ recebe mensagem e imprime em tela:
         User updated: **UID**
   3.2. Caso negativo, o servidor **SU** verifica se a quantidade máxima de usuários foi alcançada.
      3.2.1. Caso positivo, o servidor **SU** responde mensagem de erro **ERROR(17)** para cliente $\mathbf{C_i}$.
         3.2.1.1. Cliente $\mathbf{C_i}$ recebe mensagem **ERROR(17)** do servidor **SU** e imprime em tela a descrição do código de erro correspondente.
      3.2.2. Caso negativo, o servidor **SU** guarda o novo usuário no seu banco de dados e responde o cliente $\mathbf{C_i}$ com uma mensagem **OK(02)**.
         3.2.2.1. O cliente $\mathbf{C_i}$ recebe mensagem e imprime em tela:
         New user added: **UID**

2) Consultar informações de localização para um usuário (IC -> SL)

1. Um cliente $\mathbf{C_i}$ (IC) recebe comando via teclado
   find **UID**
   para consultar a localização de um usuário de id **UID**. Para isso, o cliente $\mathbf{C_i}$ envia a mensagem **REQ_USRLOC(UID)** para o servidor **SL**.
2. ※ Servidor **SL** recebe a solicitação e imprime em tela REQ_USRLOC **UID**
3. O servidor **SL** verifica se possui a localização do usuário de id **UID**.
   3.1. Em caso negativo, o servidor **SL** responde mensagem de erro **ERROR(18)** para cliente $\mathbf{C_i}$.
      3.1.1. Cliente $\mathbf{C_i}$ recebe mensagem **ERROR(18)** do servidor **SL** e imprime em tela a descrição do código de erro correspondente.
   3.2. Em caso positivo, o servidor **SL** responde mensagem **RES_USRLOC(LocId)** para cliente $\mathbf{C_i}$.
      3.2.1. O cliente $\mathbf{C_i}$ recebe mensagem e imprime em tela:
         Current location: **LocId**

3) Solicita entrada e saída de pessoa (IC -> SU -> SL)

1. Um cliente $\mathbf{C_i}$ recebe um dos comandos a seguir via teclado
   in **UID**
   out **UID**
   para registrar a entrada ou a saída de um usuário, respectivamente. Para isso, o cliente $\mathbf{C_i}$ envia a mensagem **REQ_USRACCESS(UID, direction)** para o servidor **SU**, onde direction é *in* ou *out*.
2. ※ Servidor **SU** recebe a solicitação e imprime em tela REQ_USRACCESS **UID direction**

3. O servidor **SU** verifica se o usuário **UID** existe no seu banco de dados.
    3.1. Em caso negativo, o servidor **SU** responde mensagem de erro **ERROR(18)** para cliente $\mathbf{C_i}$.
        3.1.1. Cliente $\mathbf{C_i}$ recebe mensagem **ERROR(18)** do servidor **SL** e imprime em tela a descrição do código de erro correspondente
    3.2. Em caso positivo, o servidor **SU** envia a mensagem **REQ_LOCREG(UID, LocId)** para o servidor **SL**, onde LocId deve ser **-1** caso a pessoa esteja **saindo** da catraca, ou um **número inteiro entre 1 e 10** caso esteja **entrando**.
        3.2.1. ※ O servidor **SL** recebe a mensagem e imprime em tela REQ_LOCREG **UID LocId**
        3.2.2. O servidor **SL** verifica se possui um registro de localização atual do usuário informado.
            3.2.2.1. Caso negativo, salva **LocId** como a localização atual do usuário **UID** e responde **RES_LOCREG(-1)** para o servidor **SU**.
                3.2.2.1.1. Continua como no item 2.2.1.2.1.
            3.2.2.2. Caso positivo, salva **LocId** como a localização atual do usuário **UID** e responde **RES_LOCREG(OldLocId)** para o servidor **SU**, onde **OldLoc** é a última localização salva do usuário.
                3.2.2.2.1. O servidor **SU** recebe a mensagem **RES_LOCREG(LocId)** e então responde ao cliente $\mathbf{C_i}$ com a mensagem **RES_USRACCESS(LocId)**.
                    3.2.2.2.1.1. O cliente $\mathbf{C_i}$ recebe mensagem e imprime em tela:
                    Ok. Last location: **LocId**

4) Solicitar lista de pessoas presentes em determinada localização (IC -> SL -> SU)

1. Um cliente $\mathbf{C_i}$ recebe comando via teclado
    inspect **UID LocId**
    para consultar a lista de pessoas que estão localizadas no prédio **LocId**, se autenticando como o usuário **UID**. Para isso, o cliente $\mathbf{C_i}$ envia a mensagem **REQ_LOCLIST(UID, LocId)** para o servidor **SL**.
2. ※ Servidor **SL** recebe a solicitação e imprime em tela REQ_LOCLIST **UID LocId**
3. O servidor **SL** envia uma mensagem **REQ_USRAUTH(UID)** para o servidor **SU** para verificar se o usuário **UID** tem permissão adequada.
    3.1. ※ O servidor **SU** recebe a mensagem e imprime em tela REQ_USRAUTH **UID**
    3.2. O servidor **SU** verifica se o usuário **UID** possui permissão especial.
        3.2.1. Caso negativo, o servidor **SU** responde **RES_USRAUTH(0)** para o servidor **SL**.
            3.2.1.1. O servidor **SL** recebe a mensagem **RES_USRAUTH(0)** e então responde ao cliente $\mathbf{C_i}$ com a mensagem de erro **ERROR(19)**.
                3.2.1.1.1. Cliente $\mathbf{C_i}$ recebe mensagem **ERROR(19)** do servidor **SL** e imprime em tela a descrição do código de erro correspondente
        3.2.2. Caso positivo, o servidor **SU** responde **RES_USRAUTH(1)** para o servidor **SL**.
            3.2.2.1. O servidor **SL** recebe a mensagem **RES_USRAUTH(1)** e então responde ao cliente $\mathbf{C_i}$ com a mensagem **RES_LOCLIST(UIDs)**, onde **UIDs** é a lista de UID de usuários que estão atualmente em **LocId**. Esta lista pode ser vazia.
                3.2.2.1.1. O cliente $\mathbf{C_i}$ recebe mensagem e imprime em tela (UIDs devem ser separados por vírgula seguido por espaço, isto é, ", "):
                List of people at the specified location: **UIDs**

Exemplo de saída para a lista de UIDs:
List of people at the specified location: 2023548752, 0123456789, 0024549845, 2021325487

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 o servidor e vice-versa ou o servidor tem que enviar a mensagem para outro servidor, de modo que o correto entendimento da mensagem deve ser feito por todos os programas). Caso algum dos programas receba um comando inválido, o tratamento deverá ser feito localmente, imprimindo uma mensagem de erro e voltando a estar disponível para novos comandos.

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 SU** e outro para se comunicar com o **servidor SL**, independente de quantos outros programas se comunicarem com aquele processo. Já os servidores inicializam-se com **dois** socket cada, um para a conexão com o servidor peer e um para a conexão com os clientes, e a 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**. O **servidor de usuários (SU)** deve suportar até **30 pessoas cadastrados**, enquanto o **servidor de localização (SL)** deve suportar as **localizações -1 e de 1 a 10**. 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:

- Cada mensagem possui no máximo 500 bytes.
- Para identificar as mensagens, use o código de identificação correspondente a elas. É seguro armazenar este código em um *unsigned char*.
- Para melhorar a organização do código, é **sugerido** incluir no arquivo *common.h* um conjunto de *defines* para relacionar o tipo textual das mensagens ao seu respectivo código identificador. Um arquivo pré-feito com essas definições pode ser acessado neste link.

Comandos da entrada padrão

- **Servidor**
  - ➢ **kill:** Requisita o fechamento da comunicação peer-2-peer e encerramento do servidor
- **Cliente**
  - ➢ **kill:** Requisita o fechamento das conexões com os servidores e encerramento do cliente
  - ➢ **add UID IS_SPECIAL**: Requisita a criação de usuário no servidor de usuários.
    - UID: string de exatamente 10 caracteres
    - IS_SPECIAL: 1 ou 0 (usuário tem ou não tem permissão especial, respectivamente)
  - ➢ **find UID**: Requisita a localização de um usuário no servidor de localização
    - UID: string de exatamente 10 caracteres
  - ➢ **in UID**: Requisita a entrada de um usuário em um prédio
    - UID: string de exatamente 10 caracteres
  - ➢ **out UID**: Requisita a saída de um usuário em um prédio
    - UID: string de exatamente 10 caracteres
  - ➢ **inspect UID LOC**: Requisa a lista de pessoas que estão em um prédio, com autenticação
    - UID: string de exatamente 10 caracteres a ser usada para autenticação

- LOC: número indicando o prédio desejado

Execução

Seu servidor deve receber dois números de porta na linha de comando, especificando em qual porta ele vai **`estabelecer a conexão peer-2-peer`** e em qual vai **`receber conexões dos clientes`**. Para padronização do trabalho, utilize a porta **40000** para a conexão peer-2-peer e as portas **50000** e **60000** para receber conexões de clientes nos servidores de usuários e de localização, respectivamente. Seu cliente deve receber, **estritamente nessa ordem**, o endereço IP (IPv4 ou IPv6) dos servidores, a porta do servidor de usuários, a porta do servidor de localização e o código da localização que ele representa (valor de 1 a 10). Para realizar múltiplas conexões de clientes com os servidores, basta executar múltiplas vezes o código do programa cliente.

A seguir, um exemplo de execução de dois clientes conectados com os dois servidores em quatro terminais distintos:

```
Terminal 1: ./server 40000 50000
Terminal 2: ./server 40000 60000
Terminal 3: ./client 127.0.0.1 50000 60000 1
Terminal 4: ./client ::1 50000 60000 2
```

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

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 através dos dados trocados pela rede e serão executados principalmente através do envio de comandos pela entrada padrão e subsequente verificação do resultado na saída padrão. Por essa razão, recomendamos que as impressões adicionais feitas na tela (ex.: informações de depuração) sejam suprimidas do envio final ou impressas na saída de erro dos programas (ao invés de usar a saída padrão).

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

- Abertura de comunicação cliente-servidor: **+2 pontos**
- Fechamento de comunicação cliente-servidor: **+2 pontos**
- Abertura de comunicação servidor-servidor: **+4 pontos**
- Fechamento de comunicação servidor-servidor: **+4 pontos**
- Cadastro de pessoa: **+2 pontos**
- Consulta de localização de pessoa: **+2 pontos**
- Entrada e saída de pessoa: **+6 pontos**
- Consulta das pessoas em uma certa localização: **+6 pontos**

Total: **28 pontos (80%)** + 7 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 14 pontos, visto que serão executados apenas metade dos possíveis testes.

Entrega parcial

Para auxiliar no fluxo de execução do trabalho, ele foi dividido em duas entregas. Para a entrega parcial, cada aluno deve entregar o código desenvolvido (ver seção “Formato da submissão”) e relatório em PDF de até 1 página, utilizando fonte tamanho 11, contendo o seu progresso atual no trabalho e as maiores dificuldades enfrentadas até o momento, caso tenha tido alguma. Como sugestão, informe o seu progresso no trabalho atribuindo um status (“Não iniciado”, “Em andamento” ou "Concluído”) para cada um dos pontos de avaliação citados na seção anterior.

Nesta etapa do trabalho espera-se que o aluno tenha concluído, **pelo menos**, os seguintes itens:

- Recebimento dos argumentos de forma adequada pela linha de comando
    - OBS.: Implemente a leitura dos argumentos conforme o especificado, mesmo que alguns destes argumentos ainda não estejam sendo utilizados pelo seus programas.
- Abertura de conexão entre os clientes IC e os servidores de usuários e de localização (IPv4 e IPv6)
- Fechamento de conexão com os servidores de usuários e de localização
- Cadastro de pessoa
- Consulta de localização de pessoa
    - OBS.: Como o registo de entrada e saída de pessoas ainda não estará necessariamente implementado, é esperado que o servidor de localização responda ERROR(18) ao consultar a localização de qualquer pessoa. Para testar a mensagem de sucesso, é sugerido iniciar este servidor já com alguma informação em seu banco de dados de localização.

Entrega final

Cada aluno deve entregar o código desenvolvido (ver seção “Formato da submissão”) e documentação em PDF de até 6 páginas, sem capa, utilizando fonte tamanho 11, 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, e 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 (**+7 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 documentação) a nota final será zero.**

Formato da submissão

Para cada uma das duas entregas deste trabalho, cada aluno deve submeter, além do relatório ou documentação, o **código-fonte em C** e um **Makefile** para compilação dos programas seguindo as instruções:

- 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:
    - Para a entrega parcial: TP1_PARCIAL_MATRICULA.zip
    - Para a entrega final: TP1_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 dois 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 [TP1-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 podem ser vistos em: Exemplos de Execução - TP-TW-2024-2-JG.docx

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.