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

**Execução**: Individual

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

Introdução Protocolo

Especificação das mensagens Fluxo das mensagens de controle Descrição das funcionalidades

Implementação

Comandos da entrada padrão Execução

Avaliação

Entrega parcial Entrega final Formato da entrega Prazo de entrega

Pedidos de revisão de nota Dicas e cuidados

Exemplos de execução Lembretes

**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.

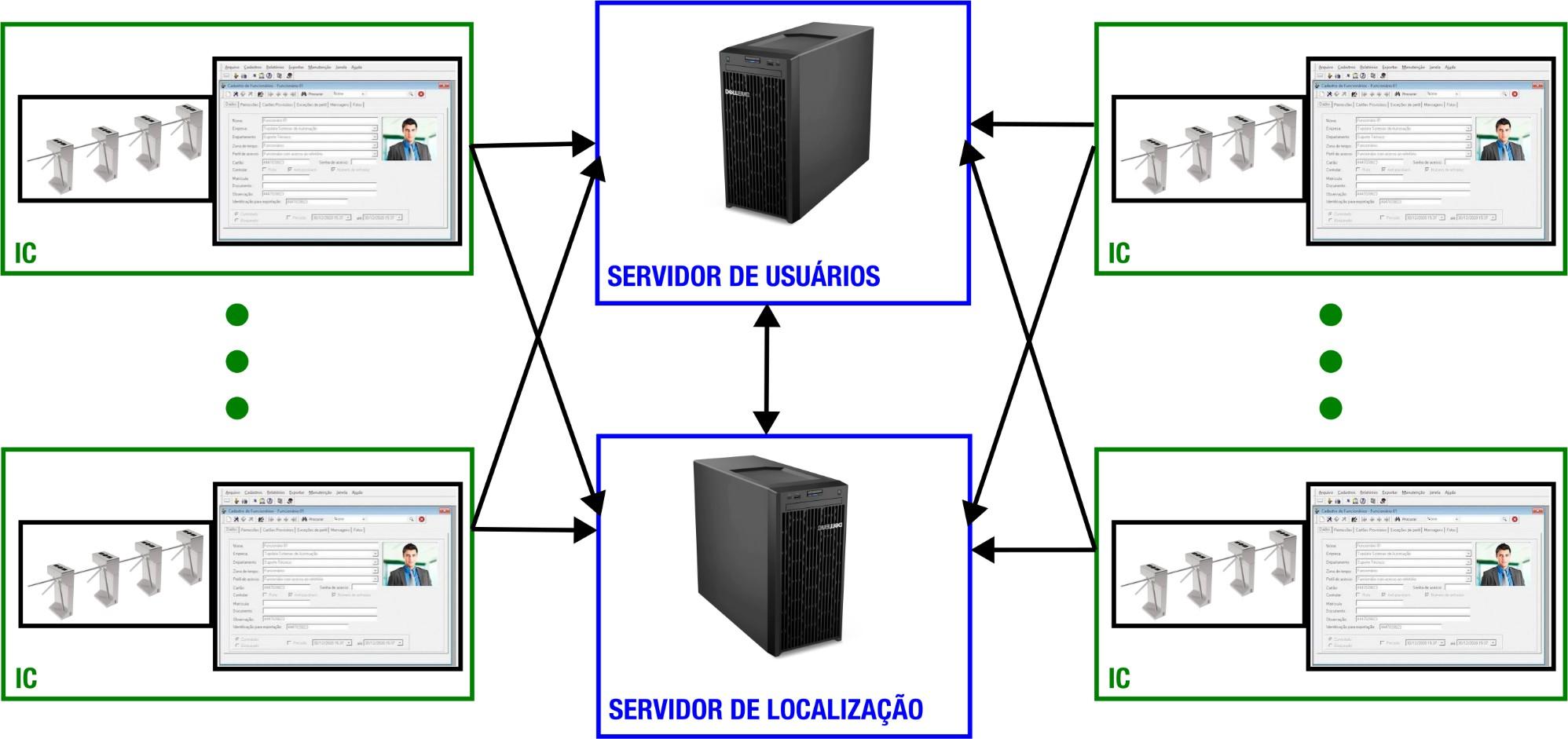


Figura 1 - Exemplo de comunicação entre as entidades do sistema. A direção da seta indica o modo de conexão: a origem se conecta ativamente ao destino. Entre os servidores há uma seta dupla para indicar que a conexão é Peer-to-Peer.

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 (Si -> Sj) | 17 | – | Mensagem de requisição de conexão entre peers |
| RES\_CONNPEER (Sj -> Si) | 18 | PidSi | Mensagem de resposta de conexão entre peers |
| REQ\_DISCPEER (Si -> Sj) | 19 | PidSi | Mensagem de requisição de encerramento de conexão entre peers |
| REQ\_CONN (Ci -> Si) | 20 | LocId | Mensagem de requisição de entrada de cliente na rede |
| RES\_CONN (Si -> Ci) | 21 | IdCi | Mensagem de resposta de entrada de cliente na rede, com a identificação IdCi do cliente Ci |
| REQ\_DISC (Ci -> Si) | 22 | IdCi | Mensagem de requisição de saída de cliente na rede, onde IdCi corresponde à identificação do cliente solicitante |

| **Mensagens de Dados** | | | |
| --- | --- | --- | --- |
| **Tipo e (Direção)** | **Código** | **Payload** | **Descrição** |
| REQ\_USRADD (Ci -> SU) | 33 | UID, is\_special | Mensagem de requisição de cadastro de pessoa |

| REQ\_USRACCESS (Ci -> SU) | 34 | UID, direction | Mensagem de requisição de entrada e saída de pessoas (direction = *in* / *out*) |
| --- | --- | --- | --- |
| RES\_USRACCESS (SU -> Ci) | 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 (Ci -> SL) | 38 | UID | Mensagem de requisição de informação sobre a localização de uma pessoa |
| RES\_USRLOC (SL -> Ci) | 39 | LocId | Mensagem de resposta de informação sobre a localização de uma pessoa |
| REQ\_LOCLIST (Ci -> 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 -> Ci) | 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 Ci. 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 Ci. 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 Si** tenta solicitar ao **servidor Sj** a abertura de comunicação por meio da mensagem

#### REQ\_CONNPEER()

* 1. Caso não haja um servidor **Sj** aberto para conexão, o servidor **Si** imprime em tela a mensagem *”No peer found, starting to listen...”* e começa a ouvir potenciais novas conexões.
  2. Caso haja um servidor **Sj** aberto para conexão, ele recebe a requisição de **Si** e verifica se a quantidade máxima de conexões peer-2-peer foi alcançada.
     1. Em caso positivo, o servidor **Sj** responde uma mensagem de **ERROR(01)** para **Si**
        1. Servidor **Si** recebe a mensagem **ERROR(01)** e imprime na tela a descrição do código de erro correspondente (vide *Especificação das Mensagens*).
     2. Em caso negativo, servidor **Sj** define um identificador **PidSi** para **Si**, registra o identificador em sua base de dados, imprime em tela a mensagem *”Peer PidSi connected”* e envia para **Si** o identificador **PidSi** definido através da mensagem **RES\_CONNPEER(PidSi).**
        1. **Si** recebe o seu identificador através da mensagem **RES\_CONNPEER(PidSi),** imprime na tela *”New Peer ID: PidSi”* define um identificador **PidSj** para o servidor **Sj**, imprime em tela a mensagem *”Peer PidSj connected”* e envia para **Sj** o identificador através da mensagem **RES\_CONNPEER(PidSj)**
        2. **Sj** recebe o seu identificador através da mensagem **RES\_CONNPEER(PidSj),**

imprime na tela *”New Peer ID: PidSj”*.

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

1. O **servidor Si** recebe comando via teclado

*kill*

e solicita ao **servidor Sj** o fechamento de comunicação através da mensagem

#### REQ\_DISCPEER(PidSi).

1. **Servidor Sj** recebe **REQ\_DISCPEER(PidSi)** e verifica se **PidSi** é o identificador do peer conectado.
   1. Em caso negativo, o servidor **Sj** responde mensagem de erro **ERROR(02)** para o servidor **Si** .
      1. Servidor **Si** recebe mensagem de **ERROR(02)** de **Sj** e imprime na tela a descrição do código de erro correspondente (vide *Especificação das Mensagens).*
   2. Em caso positivo, o servidor **Sj** remove **Si** da sua base de dados, responde mensagem

**OK(01)** para **Si** e imprime na tela a mensagem *Peer PidSi disconnected*.

* + 1. **Servidor Si** recebe mensagem **OK(01)** de confirmação, imprime em tela a descrição da mensagem, remove **Sj** da sua base de dados, imprime na tela a mensagem *Peer PidSj disconnected* e encerra a sua execução.
    2. **Servidor Sj** 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 **Ci** é iniciado para uma localização **LocId** e verifica se **LocId** é um valor entre **1 e 10**, inclusive.
   1. Caso negativo, **Ci** imprime na tela a mensagem *Invalid argument* e encerra a sua execução sem trocar nenhuma mensagem na rede.
   2. Caso positivo, **Ci** continua a execução de acordo com o passo 2.
2. O cliente **Ci** 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 **Ci** e verifica se quantidade máxima de conexões foi alcançada.
   1. Em caso positivo, os servidores **SU** e **SL** imprimem a mensagem de erro **ERROR(09)** (vide

*Especificação das Mensagens*).

* + 1. Cliente **Ci** imprime em tela descrição do código de erro **ERROR(09)**.
  1. Em caso negativo, cliente **Ci** envia a mensagem **REQ\_CONN(LocId)** para os servidores **SU** e

#### SL.

* + 1. Os servidores **SU** e **SL** definem um identificador **IdCi** para **Ci** único entre os seus clientes, registra o identificador em sua base de dados, imprime em tela a mensagem “*Client IdCi added (Loc* ***LocId****)*” e envia para o cliente **Ci** a mensagem **RES\_CONN(IdCi).** O cliente **Ci**, ao receber as mensagens **RES\_CONN(IdCi)** 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 **Ci** (IC) recebe comando via teclado

kill

e solicita aos servidores **SU** e **SL** o fechamento da comunicação por meio da mensagem

#### REQ\_DISC(IdCi).

1. Os servidores **SU** e **SL** recebem **REQ\_DISC(IdCi)** e verificam se **IdCi** existe na base de dados.
   1. Em caso negativo, os servidores **SU** e **SL** respondem mensagem de erro **ERROR(10)** para cliente **Ci**.
      1. Cliente **Ci** 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. Em caso positivo, os servidores **SU** e **SL** removem **Ci** da base de dados, respondem mensagem **OK(01)** para **Ci**, desconectam **Ci**, e imprimem em tela a mensagem

*Client* ***IdCi*** *removed (Loc* ***LocId****)*

* + 1. Cliente **Ci** 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.

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

* 1. Um cliente **Ci** (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 **Ci** envia a mensagem **REQ\_USRADD(UID, IS\_SPECIAL)** para o servidor **SU**.

* 1. ※ Servidor **SU** recebe a requisição de **Ci** e imprime em tela REQ\_USRADD **UID IS\_SPECIAL**
  2. O servidor **SU** verifica se o usuário de id **UID** já existe.
     1. Caso positivo, o servidor **SU** atualiza o valor de **IS\_SPECIAL** deste usuário e responde **Ci** com uma mensagem **OK(03)**.
        1. O cliente **Ci** recebe mensagem e imprime em tela:

User updated: **UID**

* + 1. Caso negativo, o servidor **SU** verifica se a quantidade máxima de usuários foi alcançada.
       1. Caso positivo, o servidor **SU** responde mensagem de erro **ERROR(17)** para cliente **Ci**.
          1. Cliente **Ci** recebe mensagem **ERROR(17)** do servidor **SU** e imprime em tela a descrição do código de erro correspondente.
       2. Caso negativo, o servidor **SU** guarda o novo usuário no seu banco de dados e responde o cliente **Ci** com uma mensagem **OK(02)**.
          1. O cliente **Ci** recebe mensagem e imprime em tela:

New user added: **UID**

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

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

find **UID**

para consultar a localização de um usuário de id **UID**. Para isso, o cliente **Ci** envia a mensagem

**REQ\_USRLOC(UID)** para o servidor **SL**.

* 1. ※ Servidor **SL** recebe a solicitação e imprime em tela REQ\_USRLOC **UID**
  2. O servidor **SL** verifica se possui a localização do usuário de id **UID**.
     1. Em caso negativo, o servidor **SL** responde mensagem de erro **ERROR(18)** para cliente **Ci**.
        1. Cliente **Ci** recebe mensagem **ERROR(18)** do servidor **SL** e imprime em tela a descrição do código de erro correspondente.
     2. Em caso positivo, o servidor **SL** responde mensagem **RES\_USRLOC(LocId)** para cliente **Ci**.
        1. O cliente **Ci** recebe mensagem e imprime em tela:

Current location: **LocId**

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

* 1. Um cliente **Ci** 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 **Ci** envia a mensagem **REQ\_USRACCESS(UID, direction)** para o servidor **SU**, onde direction é *in* ou *out*.

* 1. ※ Servidor **SU** recebe a solicitação e imprime em tela REQ\_USRACCESS **UID direction**
  2. O servidor **SU** verifica se o usuário **UID** existe no seu banco de dados.
     1. Em caso negativo, o servidor **SU** responde mensagem de erro **ERROR(18)** para cliente **Ci**.
        1. Cliente **Ci** recebe mensagem **ERROR(18)** do servidor **SL** e imprime em tela a descrição do código de erro correspondente
     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**.
        1. ※ O servidor **SL** recebe a mensagem e imprime em tela REQ\_LOCREG **UID LocId**
        2. O servidor **SL** verifica se possui um registro de localização atual do usuário informado.
           1. Caso negativo, salva **LocId** como a localização atual do usuário **UID** e responde **RES\_LOCREG(-1)** para o servidor **SU**.

Continua como no item 2.2.1.2.1.

* + - * 1. 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.

O servidor **SU** recebe a mensagem **RES\_LOCREG(LocId)** e então responde ao cliente **Ci** com a mensagem **RES\_USRACCESS(LocId)**.

O cliente **Ci** recebe mensagem e imprime em tela:

Ok. Last location: **LocId**

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

* 1. Um cliente **Ci** 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 **Ci** envia a mensagem **REQ\_LOCLIST(UID, LocId)** para o servidor **SL**.

* 1. ※ Servidor **SL** recebe a solicitação e imprime em tela REQ\_LOCLIST **UID LocId**
  2. O servidor **SL** envia uma mensagem **REQ\_USRAUTH(UID)** para o servidor **SU** para verificar se o usuário **UID** tem permissão adequada.
     1. ※ O servidor **SU** recebe a mensagem e imprime em tela REQ\_USRAUTH **UID**
     2. O servidor **SU** verifica se o usuário **UID** possui permissão especial.
        1. Caso negativo, o servidor **SU** responde **RES\_USRAUTH(0)** para o servidor **SL**.
           1. O servidor **SL** recebe a mensagem **RES\_USRAUTH(0)** e então responde ao cliente **Ci** com a mensagem de erro **ERROR(19)**.

Cliente **Ci** recebe mensagem **ERROR(19)** do servidor **SL** e imprime em tela a descrição do código de erro correspondente

* + - 1. Caso positivo, o servidor **SU** responde **RES\_USRAUTH(1)** para o servidor **SL**.
         1. O servidor **SL** recebe a mensagem **RES\_USRAUTH(1)** e então responde ao cliente **Ci** 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.

O cliente **Ci** 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](https://chat.nanocomp.dcc.ufmg.br/files/hjjg1xg7m7gq3jk5jp37ezjpja/public?h=MRhj4exna1rzGJjbrIaZNKJNtkOAZMih9eWwSVgvs18).

## 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

# EXEMPLOS DE EXECUÇÃO

Este documento apresenta alguns exemplos de execuções do sistema. A fim de facilitar a explicação, as tabelas a seguir detalham o passo a passo dos comandos de entrada (**em negrito**) e as informações que devem ser impressas em tela em cada instante de tempo.

## Abertura de comunicação entre Servidores(Peer-2-Peer)

A tabela 1 apresenta um cenário de comunicação entre três servidores. Como o escopo do trabalho limita o peering para apenas 2 servidores, tal controle pode ser observado no fluxo de comunicação.

| Tempo | Terminal 1 | Terminal 2 | Terminal 3 |
| --- | --- | --- | --- |
| t1 | **./server 40000 50000** |  |  |
| t2 | No peer found, starting to listen... |  |  |
| t3 |  | **./server 40000 60000** |  |
| t4 | Peer 5 connected |  |  |
| t5 |  | New Peer ID: 5 |  |
| t6 |  | Peer 9 connected |  |
| t8 | New Peer ID: 9 |  |  |
| t9 |  |  | **./server 40000 70000** |
| t10 |  |  | Peer limit exceeded |
| t11 |  |  | (Como estamos executando todos os servidores em uma mesma máquina, um erro de binding ocorrerá, pois a porta 40000 já está em uso. Neste caso, o programa deste servidor pode ser encerrado). |

*Tabela 1 - Cenário exemplo de abertura de comunicação Peer-2-Peer e controle de número máximo de conexões*

Abertura de comunicação de cliente com servidor

A tabela 2 apresenta um cenário de comunicação entre dois servidores e dois clientes. Os dois servidores realizam peering entre si, e cada cliente se conecta a ambos os servidores.

| Tempo | Terminal 1  (Peer ID: 9) | Terminal 2  (Peer ID: 5) | Terminal 3  (Client Loc: 1) | Terminal 4  (Client Loc: 7) |
| --- | --- | --- | --- | --- |
| t1 | **./server 40000 50000** |  |  |  |
| t2 | No peer found, starting to listen… |  |  |  |
| t3 |  | **./server 40000 60000** |  |  |
| t4 | Peer 5 connected |  |  |  |
| t5 |  | New Peer ID: 5 |  |  |
| t6 |  | Peer 9 connected |  |  |
| t8 | New Peer ID: 9 |  |  |  |
| t9 |  |  | **./client 127.0.0.1 50000 60000 1** |  |
| t10 | Client 2 added (Loc 1) |  |  |  |
| t11 |  | Client 14 added (Loc 1) |  |  |
| t12 |  |  | SU New ID: 2 SL New ID: 14 |  |
| t13 |  |  |  | **./client 127.0.0.1 50000 60000 7** |
| t14 | Client 3 added (Loc 7) |  |  |  |
| t15 |  | Client 15 added (Loc 7) |  |  |
| t16 |  |  |  | SU New ID: 3  SL New ID: 15 |

*Tabela 2 - Cenário de abertura de comunicação cliente-servidor em um contexto onde ocorre peering entre dois servidores.*

Fechamento de comunicação (Peer-2-peer e Cliente-servidor)

A tabela 3 apresenta um cenário de comunicação entre dois servidores e dois clientes. Os servidores e clientes já iniciam-se conectados, e essa tabela pode ser vista como uma continuação da tabela anterior. Será demonstrado o processo de encerramento de comunicação entre peers e entre cliente-servidor. A topologia é a seguinte:

Terminal 1: Servidor SU (9) Terminal 3: IC Loc 1

Terminal 2: Servidor SL (5) Terminal 4: IC Loc 7

| Tempo | Terminal 1  (Peer ID: 9) | Terminal 2  (Peer ID: 5) | Terminal 3  (Client Loc: 1) | Terminal 4  (Client Loc: 7) |
| --- | --- | --- | --- | --- |
| t1 |  |  |  | **kill** |
| t2 |  | Client 15 removed (Loc 7) |  |  |
| t3 | Client 3 removed (Loc 7) |  |  |  |
| t4 |  |  |  | SU Successful disconnect SL Successful disconnect |
| t5 |  | **kill** |  |  |
| t6 | Peer 5 disconnected |  |  |  |
| t8 |  | Successful disconnect  Peer 9 disconnected |  |  |
| t9 | No peer found, starting to listen... |  |  |  |

*Tabela 3 - Cenário de fechamento de comunicação cliente-servidor seguido de fechamento de comunicação peer-2-peer.*

Requisição de mensagem entre clientes e servidores

A Tabela 4 apresenta um cenário de comunicação entre dois servidores e dois clientes. Os servidores e clientes já iniciam-se conectados e seguem a topologia apresentada anteriormente. Para a coluna dos servidores, será denotado alguns dos pacotes recebidos (<) e enviados (>) por eles.

| Tempo | Terminal 1 (Servidor SU) | Terminal 2 (Servidor SL) | Terminal 3  (Cliente Loc 1) | Terminal 4 (Cliente Loc 7) |
| --- | --- | --- | --- | --- |
| t1 |  |  | **add 2021808080 1** |  |
| t2 | < REQ\_USRADD 2021808080 1 |  |  |  |
| t3 | > OK 2 |  |  |  |
| t4 |  |  | New user added: 2021808080 |  |
| t5 |  |  | **find 2021808080** |  |
| t6 |  | < REQ\_USRLOC |  |  |
| t7 |  | > ERROR 18 |  |  |
| t8 |  |  | User not found |  |
| t9 |  |  |  | **in 2021808080** |
| t10 | < REQ\_USRACCESS 2021808080 in |  |  |  |
| t11 |  | < REQ\_LOCREG 2021808080 7 |  |  |
| t12 | > RES\_LOCREG -1 |  |  |  |
| t13 |  |  |  | Ok. Last location: -1 |
| t14 |  |  | **find 2021808080** |  |
| t15 |  | < REQ\_USRLOC |  |  |
| t16 |  | > RES\_USRLOC 7 |  |  |
| t17 |  |  | Current location: 7 |  |
| t18 |  |  |  | **add 2021808080 0** |
| t19 | < REQ\_USRADD 2021808080 0 |  |  |  |
| t20 | > OK 3 |  |  |  |
| t21 |  |  |  | User updated: 2021808080 |
| t22 |  |  |  | **out 2021808080** |
| t23 | < REQ\_USRACCESS 2021808080 out |  |  |  |
| t24 |  | < REQ\_LOCREG 2021808080 -1 |  |  |
| t25 | > RES\_LOCREG 7 |  |  |  |
| t26 |  |  |  | Ok. Last location: 7 |
| t27 |  |  | **inspect 2021808080 -1** |  |
| t28 |  | < REQ\_LOCLIST 2021808080 -1 |  |  |
| t29 | < REQ\_USRAUTH 2021808080 |  |  |  |
| t30 |  | < RES\_USRAUTH 0 |  |  |
| t31 |  | > ERROR 19 |  |  |
| t32 |  |  | Permission denied |  |
| t33 |  |  | **add 2020909090 1** |  |
| t34 |  |  | New user added: 2020909090 |  |
| t35 |  |  | **out 2020909090** |  |
| t36 |  |  | Ok. Last location: -1 |  |
| t37 |  |  | **inspect 2020909090 -1** |  |
| t38 |  | < REQ\_LOCLIST 2020909090 -1 |  |  |
| t39 | < REQ\_USRAUTH 2020909090 |  |  |  |
| t40 |  | < RES\_USRAUTH 1 |  |  |
| t41 |  | > RES\_LOCLIST 2021808080 2020909090 |  |  |
| t42 |  |  | List of people at the specified location: 2021808080, 2020909090 |  |

*Tabela 4 - Cenário de consultas diversas entre clientes e servidores.*