

# Trabalho Prático 2 – Redes de Computadores

## Sistema de múltiplas conexões

Gilliard Gabriel Rodrigues  
Matrícula: 2019054609

Universidade Federal de Minas Gerais (UFMG)  
Belo Horizonte – MG – Brasil

[gilliard2019@ufmg.br](mailto:gilliard2019@ufmg.br)

### 1. Introdução

O problema proposto foi implementar um sistema composto por um servidor, responsável por coordenar até 15 conexões à medida em que os equipamentos entram e saem da rede, e os equipamentos (os clientes), que solicitam informações de outros equipamentos por intermédio do servidor.

O sistema permite a adição de equipamentos – fornecendo um *id* para um equipamento no momento em que ele solicita conexão –, a remoção e listagem, assim como permite que um equipamento solicite informação de outro equipamento através do servidor.

A seguir serão apresentadas as soluções e discutidos os principais desafios e dificuldades durante a implementação.

### 2. Arquitetura e Implementação

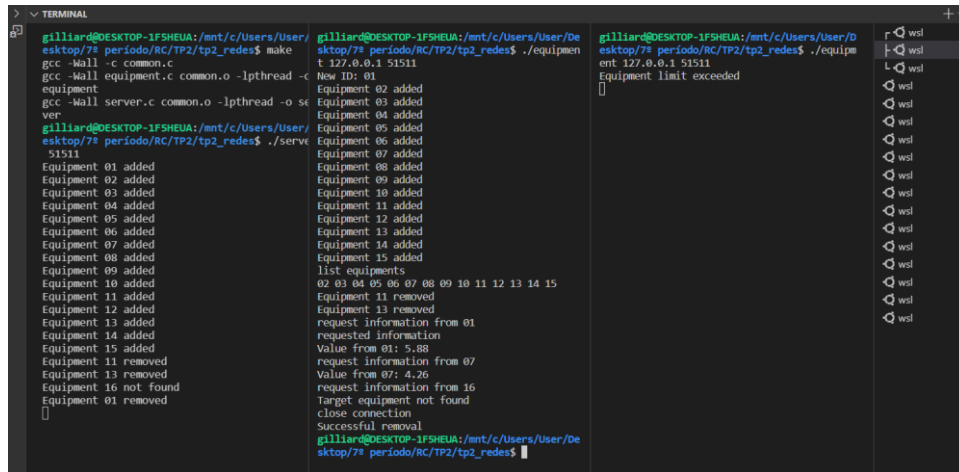
A arquitetura utilizada foi TCP/IP. No que diz respeito à base do sistema cliente-servidor, a lógica seguida foi a mesma ensinada pelo professor Ítalo Cunha durante as aulas de Introdução à Programação em Redes, disponibilizadas no Moodle – agora seguindo a abordagem multi-thread, com as alterações necessárias para conseguir enviar uma mensagem a vários clientes e utilizando apenas o protocolo IPv4.

As alterações necessárias para tornar o sistema multi-thread foram tranquilas, pois como citado anteriormente, as aulas postadas no Moodle ajudaram bastante. No entanto, enviar uma mesma mensagem para vários clientes foi uma *feature* que demorou para ser implementada, já que era algo inédito e várias tentativas não funcionavam. Ao invés de utilizar UDP para isso, a abordagem adotada foi adaptar o TCP para realizar essa tarefa, iterando sobre uma lista de sockets e usando a função *send* para enviar a mensagem para cada um deles.

Outra dificuldade enfrentada durante o desenvolvimento foi em listar os equipamentos no cliente, uma vez que embora a função que monta a *string* com a lista de *id*'s tenha sido fácil de implementar, as alterações no arquivo *equipment.c* necessárias para que ela funcionasse corretamente se mostraram trabalhosas – demorou para perceber que seria necessário ter também uma função para manipular *threads* no cliente, parecida com a que tinha no servidor e que foi ensinada na playlist de vídeos. Fora essas dificuldades, vale mencionar o considerável tempo gasto corrigindo erros de compilação e trechos escritos invertidos – como para testar a *feature* era necessário implementar todo o fluxo primeiro, em vários momentos passava-se muito tempo apenas programando sem saber se iria funcionar.

A fim de gerenciar os equipamentos foi utilizado um vetor booleano, de forma que a posição *i* representasse o equipamento de *id i+1* e caso esse equipamento estivesse presente no sistema, o valor na posição *i* seria *true*. Os sockets eram armazenados em um vetor de inteiros, tal que a posição *i* continha o socket cujo *id* é *i+1*. Tanto os vetores quanto um contador relativo à quantidade de equipamentos disponíveis no sistema são variáveis globais.

Abaixo é possível ver um exemplo de execução do sistema:



As próximas seções irão tratar das principais particularidades do sistema, ou seja, as implementações das operações – adicionar equipamento, remover equipamento, listar equipamentos e consultar informação de um equipamento – e os respectivos tratamentos de erros. Essas implementações foram feitas no arquivo *server.c*, que também conta com uma função que avalia o tipo de comando a partir da entrada no terminal, chamando a função correta para cada caso (listar, remover ou consultar informação).

## 2.1 Adicionar equipamento

A primeira função que implementa uma das quatro operações chama-se *adicionar\_equipamento*, que recebe um vetor de caracteres representando o buffer e um inteiro representando o socket do equipamento. Primeiro é verificado se o número máximo de equipamentos já foi alcançado e, se for o caso, envia como mensagem para o equipamento a string “07,-, -, 04”, que será interpretada pelo equipamento de forma que ele exiba a mensagem “Equipment limit exceeded”. Caso haja espaço para mais um equipamento, o algoritmo verifica se o equipamento já existe e, se não existir, o adiciona, imprime no servidor uma mensagem falando que o equipamento foi adicionado, envia ao equipamento a string “03,-,-, *idEQ<sub>i</sub>*” – em que *idEQ<sub>i</sub>* é o *id* do equipamento que foi adicionado –, que será interpretada pelo equipamento de forma que a mensagem “New ID: *idEQ<sub>i</sub>*” seja impressa no equipamento. Por fim, se o equipamento tiver sido adicionado, a mensagem “01, *idEQ<sub>i</sub>*,-,-” será enviada a todos os outros equipamentos através da função *enviar\_broadcast* e essa mensagem fará com que eles imprimam em suas telas “Equipment *idEQ<sub>i</sub>* added”.

## 2.2 Remover equipamento

A estrutura da função relativa à operação de remover equipamento, cujo nome é *remover\_equipamento*, é semelhante à da função *adicionar\_equipamento*. As principais diferenças são que ao invés de *settar* como *true* a posição relativa ao equipamento em questão e incrementar o contador de sensores disponíveis, ocorre o oposto. Assim como ao invés de lançar a mensagem de erro referente ao número máximo de equipamentos ter sido atingido, é lançada a mensagem referente a algum equipamento que não existe no sistema. Dado que o equipamento exista e tenha sido removido, o servidor imprime a mensagem “Equipment *idEQ<sub>i</sub>* removed” e é enviado uma mensagem para todos os outros equipamentos, de forma que eles também imprimam essa mensagem notificando a remoção do equipamento.

## 2.3 Listar equipamentos

A função relativa à operação de listar sensores de um equipamento chama-se *listar\_equipamentos* e é bem simples também. O algoritmo itera no vetor de equipamentos, concatenando os *id*'s dos equipamentos numa *string* e concatenando ela ao buffer no final. Conforme explicado na seção anterior, essa *feature* se mostrou problemática porque precisava de alterações no arquivo *equipment.c* que não haviam sido percebidas inicialmente.

## 2.3 Consultar informação

A função relativa à operação que permite que um equipamento recupere informação de outro equipamento chama-se *requisitar\_informacao\_equipamento*. Aqui é validado tanto se o equipamento solicitante quanto

o equipamento requisitado existem e, caso não existam, são enviadas as mensagens de erro relativas a cada caso. Caso o algoritmo passe por essas validações, a string “05,idEQ<sub>i</sub>,idEQ<sub>j</sub>,-” é enviada para o equipamento requisitado, que exibirá a mensagem “*requested information*” no terminal e a função *enviar\_informacao Equipamento* – que gera a leitura do equipamento e coloca no buffer – é chamada.

### **3. Conclusão**

Os pontos mais importantes para conseguir resolver o trabalho foram, sem dúvidas, aprender a aperfeiçoar o sistema de forma que ele suportasse multi-thread e aprender a enviar uma mensagem para vários clientes, pois esses são os pontos-chave do TP, aprendizado que foi, em partes, facilitado pela playlist disponibilizada para os alunos.

Após tornar o sistema multi-thread e conseguir enviar a mesma mensagem para vários clientes, o maior desafio foi implementar a listagem de equipamentos por parte de cada cliente, sem fazer com que o cliente fechasse após o comando ser dado, assim como exibisse corretamente os dados dependendo do terminal em que ele fosse acionado. Tendo feito isso, implementar as outras funções se tornou um processo mais fácil, devido às suas similaridades.

Em suma, foi possível praticar os conceitos de programação em redes vistos em aula e consolidar o aprendizado novo.

## **Referências**

CUNHA, Ítalo. Playlist de Introdução à Programação em Redes. Departamento de Ciência da Computação. Universidade Federal de Minas Gerais. 2022.