**Universidade Católica de Santos**

Sistemas Distribuídos - Atividade Prática

# Gerenciador de Arquivos Remoto (GAR)

Santos, SP
2024

**UNIVERSIDADE CATÓLICA DE SANTOS**
**CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS**
**BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO**

# ATIVIDADE PRÁTICA 7º SEMESTRE

Planejamento do projeto - GAR - Sistemas Distribuídos

Trabalho prático apresentado ao curso de Ciência da Computação da Universidade Católica de Santos como instrumento de avaliação para a disciplina de sistemas distribuídos do 7º Semestre.

**Vitor Cordeiro Paes Prieto**

Santos
2024

**Autor do documento:**

| Nome | E-mail |
|---|---|
| Vitor Cordeiro Paes Prieto | vitor.c.prieto@unisantos.br |

**Histórico de Versões:**

| Versão | Data | Descrição |
|---|---|---|
| 0.5 | 25/03/2024 | Criação do arquivo e formatação inicial. Definição prévia do sistema. |
| 1.0 | 26/03/2024 | Inclusão de referências bibliográficas. |
| 1.5 | 23/04/2024 | Inclusão do tópico “Código-fonte” e documentação de planos futuros. |
| 2.0 | 31/05/2024 | Inclusão de testes e documentação do programa. |

## 1.0 INTRODUÇÃO

Atualmente existem diversos programas e plataformas que, de uma maneira ou outra, realizam transferência e armazenamento de arquivos, seja ela local, remota ou de maneira híbrida, como o Google Drive, Dropbox, Microsoft OneDrive, etc. Diante disso, o presente documento propõe-se a descrever um simples sistema gerenciador de arquivos remotos (GAR) com a utilização de programação em sockets, que representa o projeto escolhido para o desenvolvimento da atividade prática proposta pela disciplina de Sistemas Distribuídos, ministrada pelo Professor Mestre Guilherme Apolinário Silva Novaes, durante o 7° semestre do curso de Ciência da Computação da Universidade Católica de Santos.

O desenvolvimento do programa GAR não oferece inovações às tecnologias já conhecidas e visa apenas o uso acadêmico das ideias estudadas, apresentando-se como um simples e primitivo teste de conceito, o qual alicerça-se somente na vontade de desvendar o intrínseco e interessante campo de conhecimento de sistemas distribuídos e programação paralela com soquetes em linguagens como C e C++.

## 2.0 OBJETIVOS GERAIS

Como supramencionado, o projeto GAR objetiva ser uma prova de conceito para os temas abordados em aula (soquetes e programação paralela), para solidificar o conhecimento adquirido. De maneira geral, o programa idealizado deverá utilizar um computador e remotamente manipular documentos de uma segunda máquina.

## 2.1 OBJETIVOS ESPECÍFICOS

Implementar duas conexões por sockets (máquinas com sistema operacional Windows), uma para o envio de comandos e outra para a retirada das informações processadas, e, para isso, criar e implementar os comandos “criptografar”, “enviar” e “deletar” no computador servidor para manipular dados do computador cliente de maneira remota (recebendo o nome do arquivo-alvo como entrada).

## 3.0 REQUISITOS

Nos seguintes tópicos, esboços dos requisitos do projeto serão apresentados. É importante ressaltar que o presente trabalho encontra-se em desenvolvimento e o atual documento pode não representar o programa final com total acurácia; novos requisitos podem surgir e antigos serem descartados, como por exemplo a criptografia de mensagens provenientes do servidor. Atualmente o conceito principal do projeto é trabalhar com duas conexões distintas, mas caso testes com apenas uma conexão se mostrem promissores, atualizaremos este documento para refletir a realidade do desenvolvimento.

### 3.1 REQUISITOS FUNCIONAIS

| [RF-01] Conexão socket cliente-servidor | |
|---|---|
| Prioridade | Essencial |
| Descrição | Conexão que possibilita o envio de informações da segunda máquina (cliente) para a primeira (servidor). |
| Pré-condições | N/A |
| Requisitos não-funcionais relacionados | [RNF-02], [RNF-03] |

| [RF-02] Conexão socket servidor-cliente | |
|---|---|
| Prioridade | Essencial |
| Descrição | Conexão que possibilita o envio de comandos da primeira máquina (servidor) para a segunda (cliente). |
| Pré-condições | N/A |
| Requisitos não-funcionais relacionados | [RNF-01], [RNF-02], [RNF-03] |

| [RF-03] Envio de arquivos | |
|---|---|
| Prioridade | Essencial |
| Descrição | Comando de transferência de arquivos originário do servidor para receber arquivos contidos no cliente. |
| Pré-condições | [RF-01], [RF-02] |
| Requisitos não-funcionais relacionados | [RNF-01], [RNF-02], [RNF-03] |

| [RF-04] Deleção | |
|---|---|
| Prioridade | Importante |
| Descrição | Comando de deleção originário do servidor para deletar arquivos contidos no cliente. |
| Pré-condições | [RF-02] |

| | |
|---|---|
| Requisitos não-funcionais relacionados | [RNF-01], [RNF-02], [RNF-03] |

| [RF-05] Criptografia | |
|---|---|
| Prioridade | Importante |
| Descrição | Comando de criptografia originário do servidor para cifrar arquivos contidos no cliente. |
| Pré-condições | [RF-02] |
| Requisitos não-funcionais relacionados | [RNF-01], [RNF-02], [RNF-03] |

| [RF-06] Modo de Espera (tratativa de exceções) | |
|---|---|
| Prioridade | Muito importante |
| Descrição | Caso o servidor seja desconectado, o cliente deve continuar em modo de espera (stand-by), verificando de 15 em 15 segundos se há conexão com o servidor novamente. |
| Pré-condições | [RF-02] |
| Requisitos não-funcionais relacionados | [RNF-02], [RNF-03] |

| [RF-07] Desconectar | |
|---|---|
| Prioridade | Desejável |
| Descrição | Comando para desconectar o servidor e deixar o cliente em modo de espera. |
| Pré-condições | [RF-02] |
| Requisitos não-funcionais relacionados | [RNF-01], [RNF-02], [RNF-03] |

| [RF-08] Remoção do Cliente | |
|---|---|
| Prioridade | Desejável |
| Descrição | Comando para remover o programa do computador do cliente. |
| Pré-condições | [RF-02] |

| | |
|---|---|
| Requisitos não-funcionais relacionados | [RNF-01], [RNF-02], [RNF-03] |

| [RF-09] Escaneamento de diretórios | |
|---|---|
| Prioridade | Muito importante |
| Descrição | Comando para escanear os diretórios na máquina do cliente. |
| Pré-condições | [RF-01], [RF-02] |
| Requisitos não-funcionais relacionados | [RNF-01], [RNF-02], [RNF-03] |

## 3.2 REQUISITOS NÃO FUNCIONAIS

| [RNF-01] Criptografia no envio de comandos | |
|---|---|
| Descrição | Implementar um método de criptografia para as mensagens (comandos) originados do servidor com o uso de uma chave padrão (início da sessão) e a geração de uma chave única durante a execução do programa. A chave única será válida durante a sessão e uma nova deve ser criada quando a sessão se iniciar novamente. |

| [RNF-02] Auto-início | |
|---|---|
| Descrição | Automaticamente reiniciar o programa no computador do cliente quando o sistema operacional for iniciado. |

| [RNF-03] Reconhecimento de sistema operacional (tratativa prévia de exceções) | |
|---|---|
| Descrição | Reconhecer se o cliente possui uma máquina Windows e não executar o programa caso contrário. |

## 4.0 METODOLOGIA

Para o desenvolvimento do projeto GAR, serão utilizadas as linguagens de programação C e/ou C++, além da biblioteca winsock2 e as demais necessárias para realizar programação socket em máquinas com o sistema operacional Windows, além de bibliotecas para programação paralela, caso essa abordagem seja inevitável.

Atualmente, a ideia geral do projeto é dividir as duas conexões em dois códigos diferentes: um executado pelo cliente e outro executado pelo servidor. Dentro de cada arquivo-fonte, haverá uma conexão; a conexão Servidor-cliente (máquina do servidor) tem o objetivo de enviar comandos com ações que manipulam os dados do cliente e ela pode ser desativada, o que resultaria em um estado de espera por parte do cliente. A conexão Cliente-servidor (máquina do cliente), por sua vez, atua na transferência de dados para o servidor, que é realizada por comandos enviados pelo mesmo.

Além das conexões, os programas devem realizar ações secundárias, quando necessárias, as quais são descritas abaixo:

**Conexão Cliente-servidor:**

- Entrar em modo de espera quando não há a conexão Servidor-cliente;
- Ser inicializado juntamente ao sistema operacional, caso possível;
- Procurar pela conexão Servidor-cliente a cada 15 segundos;
- Escanear a árvore de diretórios do cliente e enviar esta informação ao servidor quando sua conexão for estabelecida e quando for requisitado;
- Decifrar as mensagens recebidas com o auxílio da chave enviada pelo servidor.

**Conexão Servidor-cliente:**

- Criar e enviar ao cliente uma chave criptográfica ao início de cada sessão;
- criptografar os comandos enviados utilizando a chave gerada na sessão atual.

Ademais, como referencial teórico preliminar, além das aulas ministradas pelo professor Guilherme Apolinário, o artigo “Windows Sockets 2” fornecido pela Microsoft em sua plataforma “Microsoft Learn” será utilizado durante o trabalho (disponível em: Windows Sockets 2 - Win32 apps | Microsoft Learn), juntamente às seguintes obras:

- TCP/IP Sockets in C: Practical Guide for Programmers, escrito por **Michael J. Donahoo** e **Kenneth L. Calvert**. (Este livro usa programação de sockets em Unix, então será usado apenas para estudar conceitos fundamentais e generalizados).

- Hands-On Network Programming with C: Learn socket programming in C and write secure and optimized network code, escrito por **Lewis Van Winkle**.

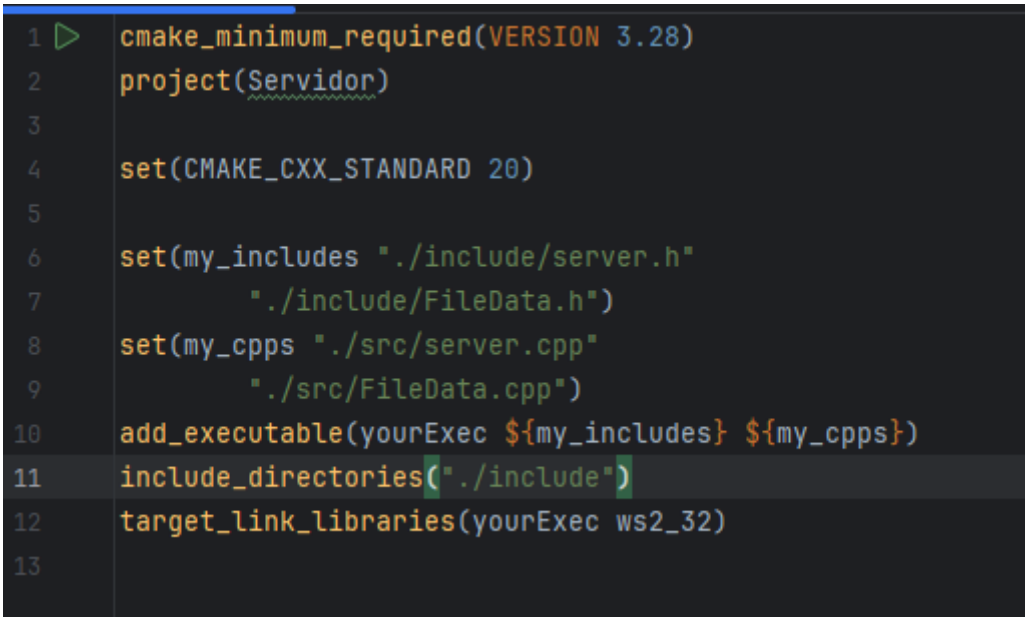
## 5.0 DESENVOLVIMENTO DO CÓDIGO-FONTE E DESAFIOS

Durante o desenvolvimento do GAR, inicialmente feio no editor de texto Visual Studio Code, migramos o código para a IDE CLION, pois houveram problemas de incompatibilidade com o uso de funções adicionadas em padrões C++ superiores ao C++ 17. Atualmente o programa se apresenta como um mínimo produto viável (versão 0.1), realizando a transferência de arquivos em computadores dentro de uma mesma rede, mas sem a implementação de todas as funcionalidades propostas, como deleção e cifração de arquivos e criptografia dos comandos enviados pelo servidor. Até o presente momento, foram implementadas apenas funções para a transferência de arquivos e para a varredura de diretórios na máquina do cliente (contando com opções pré-definidas para pastas conhecidas no Windows, como Documentos, Downloads e Desktop, e uma opção para varredura de pastas personalizadas, que são entradas no terminal do servidor).

Tratando-se do código, o mesmo foi dividido em dois projetos diferentes na IDE CLION, um para o servidor e outro para o cliente. Ambos contam com a implementação da classe FileData, que é responsável por guardar informações relacionadas ao arquivo transferido. A seguir, uma tabela com a lista de elementos incluídos na compilação por CMAKE de ambos os projetos:

| Elementos compilados por projeto | |
|---|---|
| **Cliente** | **Servidor** |
| client.cpp | server.cpp |
| client.h | server.h |
| FileData.cpp | FileData.cpp |
| FileData.h | FileData.h |

Figura 1: CMAKE para o projeto Servidor.



```
cmake_minimum_required(VERSION 3.28)
project(Servidor)

set(CMAKE_CXX_STANDARD 20)

set(my_includes "./include/server.h"
        "./include/FileData.h")
set(my_cpps "./src/server.cpp"
        "./src/FileData.cpp")
add_executable(yourExec ${my_includes} ${my_cpps})
include_directories("./include")
target_link_libraries(yourExec ws2_32)
```

Fonte: Autor.

Figura 2: CMAKE para o projeto Cliente.

```
cmake_minimum_required(VERSION 3.28)
project(Cliente)

set(CMAKE_CXX_STANDARD 20)

set(my_includes "./include/client.h"
        "./include/FileData.h")
set(my_cpps "./src/client.cpp"
        "./src/FileData.cpp")
add_executable(yourExec ${my_includes} ${my_cpps})
include_directories("./include")
target_link_libraries(yourExec ws2_32)
```

Fonte: Autor.

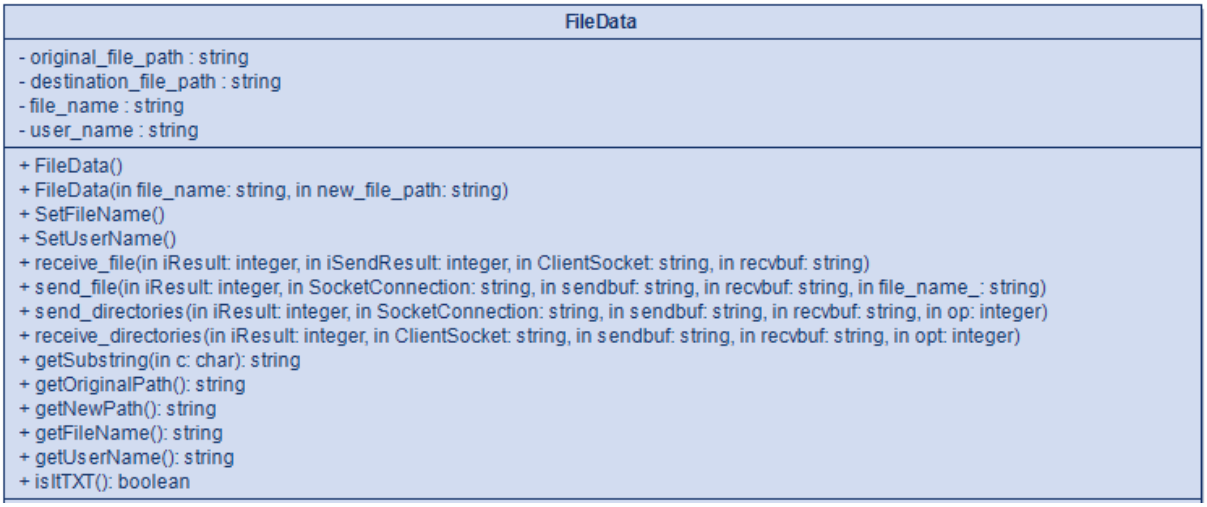
Na biblioteca Winsock2, o envio e recebimento de arquivos é feito pelas funções **send** e **recv**, que recebem como atributo a conexão socket, um ponteiro para um array de caracteres (char *) que age como um array de bytes, um inteiro com o tamanho do buffer e eventuais flags. Ambas funções retornam um inteiro (maior que zero caso a transferência seja bem sucedida). Durante o fluxo do programa, utilizamos std::strings para a manipulação de sequências de caracteres e normalmente realizamos a conversão para char * strings antes do uso de ambos **send** e **recv.**

Para a varredura de diretórios, utilizamos a biblioteca filesystem (adicionada no C++ 17); descobrimos primeiro o nome de usuário utilizado no computador e depois vasculhamos as pastas documentos, downloads, desktop e pictures por meio da função **directory_iterator**.

Após a conexão por socket é estabelecida, ambos os programas (cliente e servidor) entram em um loop, até que a opção escolhida pelo servidor seja o encerramento da conexão (opção 3). O cliente espera pelas opções 1 e 2 que vasculham diretórios e enviam arquivos, respectivamente (o número escolhido é enviado por socket para o código em execução no computador do cliente). Independentemente da escolha, a transferência de informações para o servidor também é feita com um loop, usando como parada o fim do arquivo selecionado (flag **EOF** em uma variável do tipo char) ou o recebimento de uma string específica para a varredura de diretórios, que envia o texto “END” após a última pasta ser listada.

A classe FileData, por sua vez, é responsável por guardar o antigo diretório do arquivo (no computador do cliente), o novo diretório (no computador do servidor, escolhido no próprio código-fonte), o nome do arquivo (utilizado para criar o caminho do arquivo na máquina do servidor com o novo diretório) e o nome do usuário no computador do cliente (para vasculhar os diretórios). Ademais, a classe também tem uma função para verificar se a extensão do arquivo é “.txt”, para adicionar um finalizador de string ‘\0’ ao final do arquivo quando o mesmo é transferido. A seguir, um diagrama de classes UML criado para a classe FileData, com todos os seus atributos e métodos:

Figura 3: Diagrama de classe FildeData.




Fonte: Autor.

Como o código em sua totalidade é extenso, o mesmo pode ser visualizado pela seguinte pasta compartilhada (google drive - e-mail Unisantos com acesso):
🗀 GAR - FINAL

**Observação**: Antes de executar o código, por favor atualize a linha 13 do arquivo Server.cpp com o diretório final dos arquivos transferidos.

## 6.0 CONSIDERAÇÕES FINAIS

Durante o desenvolvimento do projeto, encontramos muitas dificuldades, o que nos impossibilitou de implementar todas as funcionalidades anteriormente definidas para o GAR. Entretanto, o ganho de conhecimento ao final do processo é notório. Utilizar Winsock2 e C++ para esta atividade foi algo que incrementou a curva de aprendizado, mas nos ajudou a ter uma visão mais “manual” de conexões por socket em linguagens de programação.

## REFERÊNCIAS BIBLIOGRÁFICAS

DONAHOO, Michael J.; CALVERT, Kenneth L. **TCP/IP sockets in C: practical guide for programmers**. Morgan Kaufmann, 2009.

Microsoft. **Windows Sockets 2**. 2021. Disponível em: Windows Sockets 2 - Win32 apps | Microsoft Learn.

VAN WINKLE, Lewis. **Hands-On Network Programming with C: Learn socket programming in C and write secure and optimized network code**. Packt Publishing Ltd, 2019.