Nome: Rafaela Barbosa Trindade. **RA:** 11201921826. **Turma:** DA2.

SISTEMA DE BACKUP DISTRIBUÍDO

LINK DO PROJETO
https://github.com/RafaelaBT/Backup-Distributed-System

SUMÁRIO

1 - DESCRIÇÃO

Sistema distribuído responsável por realizar backups de arquivos com suporte a replicação de conteúdo.

2 - ESTRUTURA

O sistema é dividido em três partes, que contém:

1. **Manager** - um gerenciador.
2. **Servers** - três servidores: cada servidor possui a sua própria pasta, além de uma pasta Backup, onde os arquivos serão armazenados.
3. **Clients** - dois clientes: cada cliente possui a sua própria pasta, além de uma pasta Files, onde estão os arquivos que serão enviados para *backup*.

Ademais, cada parte possui seu arquivo de **RPC** (*Remote Procedure Call*) - RPCManager, RPCServer e RPCClient.

3 - IDEIA

O RPC possibilita um programa realizar uma chamada a um procedimento ou método em um sistema remoto como uma chamada de função local.

- O cliente chama o stub do cliente, um procedimento local que empacota os parâmetros em uma mensagem e a envia para o servidor;
- O servidor recebe a mensagem e a passa para o stub do servidor, que desempacota os parâmetros e executa o procedimento ou método solicitado;
- O mesmo processo é feito na ordem inversa com o resultado da função.

A mensagem é enviada na forma de json por TCP, com um tamanho de pacote de 65536 bytes (64 KB, que é o tamanho mais comum). Para envio de arquivos, os arquivos são quebrados e recebidos em pacotes.

O gerenciador e os servidores podem atender múltiplos clientes ao mesmo tempo através de threads.

4 - FUNCIONAMENTO

RPCManager.py

Possui uma classe chamada RPCManager que contém todas as informações do gerenciador (endereço, métodos e servidores).

__innit__(ip:texto, porta:inteiro): construtor da classe.

registerMethod(função): registra os métodos pertencentes a classe.

registerInstace(instância): registra as instâncias da classe.

run(): inicia o gerenciador. Realiza a abertura do soquete para a conexão e, assim que um servidor ou cliente se conecta, inicia uma thread para atendê-lo, enviando o soquete e o endereço. Ao final, o gerenciador fecha o soquete e espera todas as threads terminarem de executar.

__handle__(soquete, endereço:tupla): lida com cada uma das threads. Recebe e desempacota a mensagem que contém o nome da função e seus parâmetros. Imprime a mensagem recebida. Chama a função solicitada e envia seu resultado como resposta empacotando em uma mensagem.

getKey(endereço:tupla): retorna o endereço passado como um texto no formato IP:PORTA.

serverRegister(endereço:tupla, capacidade:inteiro): registra o servidor conectado ao gerenciador. Adiciona o servidor ao dicionário de servidores do gerenciador, cuja chave é o endereço no formato IP:PORTA e o valor é um dicionário com as características do servidor, como a capacidade (futuramente, também pode-se acrescentar os nomes dos arquivos do

servidor). OBS: a capacidade por enquanto se refere ao tamanho total dos arquivos armazenados no servidor em bytes.

showServers(): mostra todos os servidores conectados ao gerenciador.

getSize(): retorna a quantidade de servidores conectados ao gerenciador.

chooseServer(endereço:tupla): retorna o endereço do servidor com maior capacidade (que tem menor quantidade em bytes de arquivos armazenados). Pode receber ou não o endereço de um servidor que não deve ser escolhido.

updateCapacity(endereço:tupla, capacidade:inteiro): atualiza a capacidade do servidor no dicionário dos servidores do gerenciador.

delServer(endereço:tupla): remove o servidor do dicionário de servidores do gerenciador.

manager.py

Cria uma instância da classe RPCManager, registra suas funções e inicia o gerenciador. Por padrão, o gerenciador é iniciado na porta 65432. Por segurança, as funções chamam as funções locais do RPCManager.

string(texto): retorna o texto recebido.

addServer(endereço:tupla, capacidade:inteiro): chama serverRegister.

removeServer(endereço:tupla): chama removeServer.

getServer(endereço:tupla=Vazio): chama chooseServer. OBS: pode receber ou não um endereço.

updateServer(endereço:tupla, capacidade:inteiro): chama updateServer.

getQuantity(): chama getSize.

RPCServer.py

Possui uma classe chamada RPCServer que contém todas as informações do servidor (soquete e endereço do gerenciador, endereço, capacidade e métodos do servidor).

__innit__(ip:texto, porta:inteiro, capacidade:inteiro, ip do gerenciador:texto, porta do gerenciador:inteiro): construtor da classe.

registerMethod(função): registra os métodos pertencentes a classe.

registerInstace(instância): registra as instâncias da classe.

run(): inicia o servidor. Se conecta com o gerenciador e realiza a abertura do soquete para a conexão dos clientes, assim que um cliente se conecta, inicia

uma thread para atendê-lo. Ao final, o servidor fecha o soquete e espera todas as threads terminarem de executar.

__connect__(): realiza a conexão com o gerenciador. Se conecta, atrela o soquete da conexão ao soquete atributo da instância, e invoca a função addServer do gerenciador para registrar o servidor.

__handle__(soquete, endereço:tupla): lida com cada uma das threads. Recebe e desempacota a mensagem que contém o nome da função e seus parâmetros. Imprime a mensagem recebida. Se a função solicitada for a sendFilename, envia uma mensagem de resposta, avisando que o nome do arquivo foi recebido e indicando que receberá os pacotes. Recebe e escreve os pacotes do arquivo na pasta Backup, até receber EOF (*End Of File*). Atualiza a capacidade do servidor e chama a função updateServer do gerenciador. Envia a confirmação de que o arquivo foi recebido para o cliente. Se a conexão não for de um servidor, então verifica se há servidores na rede para fazer o backup (deve haver mais de um servidor) e solicita o endereço de um servidor secundário, chamando a função getServer do gerenciador. Realiza o backup chamando a função local backup. Se a função solicitada não for uma função para enviar arquivos, então chama a função solicitada e envia seu resultado como resposta empacotando em uma mensagem.

backup(endereço:tupla, caminho:texto, nome do arquivo:texto): se conecta com o servidor secundário e envia o arquivo de backup. Chama a função sendFilename, passando o nome do arquivo como parâmetro e o texto “backup” como identificador. Envia os pacotes do arquivo e o EOF. Recebe, desempacota e imprime a mensagem recebida como resposta.

__getattr__(__nome): stub do servidor. Empacota e envia as funções remotas chamadas e seus atributos em uma mensagem. Recebe, desempacota e retorna as mensagens recebidas como resposta.

disconnect(): desconecta o servidor do gerenciador, o removendo do dicionário de servidores. Fecha o soquete da conexão.

__del__(): desconecta/deleta o servidor do gerenciador.

Nome: Rafaela Barbosa Trindade. **RA:** 11201921826. **Turma:** DA2.

server.py

Existem três servidores, cada um com sua própria porta (65433, 65434, 65435) e arquivos, localizados em Backups. Cria uma instância da classe RPCServer, registra suas funções e inicia o servidor.

getSize(caminho): retorna o valor total em bytes do tamanho dos arquivos em uma pasta. (Utilizada para obter a capacidade inicial do servidor.)

string(texto): retorna o texto recebido.

sendPath(): retorna o caminho do servidor.

RPCClient.py

Possui uma classe chamada RPCClient que contém todas as informações do cliente (soquete e endereço de conexão).

__innit__(ip:texto, porta:inteiro): construtor da classe.

connect(): realiza a conexão com o gerenciador/servidor. Atrela o soquete da conexão ao atributo soquete da instância, a fim de manter a conexão.

isConnected(): envia uma mensagem de teste para o gerenciador/servidor, a fim de verificar a conexão.

sendFile(caminho, nome do arquivo:texto): envia o arquivo para o servidor. Solicita o endereço do servidor para o gerenciador, através da função remota getServer. Cria uma nova instância cliente com o endereço do servidor recebido. Inicia a conexão com o servidor. Envia o nome do arquivo para o servidor, chamando a função remota sendFilename. Abre o arquivo para leitura e envia pacotes do arquivo. Por fim, envia EOF, recebe e desempacota a resposta do servidor e fecha a conexão, utilizando a função local disconnect. Retorna Verdadeiro ou Falso dependendo do sucesso do envio.

__getattr__(__nome): stub do cliente. Empacota e envia as funções remotas chamadas e seus atributos em uma mensagem. Recebe, desempacota e retorna as mensagens recebidas como resposta.

disconnect(): desconecta o cliente do gerenciador/servidor. Fecha o soquete da conexão.

__del__(): desconecta/deleta o cliente do gerenciador/servidor. Fecha o soquete da conexão.

Nome: Rafaela Barbosa Trindade. **RA:** 11201921826. **Turma:** DA2.

client.py

Existem dois clientes, cada um com seus próprios arquivos, localizados em Files. Todas as mensagens impressas para o cliente são de comunicação entre o cliente e o sistema. É criada uma instância da classe RPCClient e feita a conexão com o gerenciador através da função local connect do RPCClient. Enquanto o usuário ainda quiser enviar arquivos, é verificada a conexão com o gerenciador, listados os arquivos disponíveis na pasta local e recebido o nome do arquivo digitado pelo usuário. Caso o usuário digite o nome errado, será solicitado novamente. Caso o usuário digite o nome do arquivo corretamente, o caminho e nome do arquivo é enviado, chamando a função local sendFile do RPCClient. Caso o usuário tecle ENTER, então o sistema é fechado, encerrando a conexão ao chamar a função local disconnect do RPCClient.

5 - REFERÊNCIAS

TCP: Socket Programming in Python (Guide)

Multi-threading: Socket Programming with Multi-threading in Python - GeeksforGeeks

RPC: Coding Remote Procedure Call (RPC) with Python | by Taras Zherebetskyy | Medium