UFBA - Universidade Federal da Bahia
Disciplina: MATA88 - Sistemas Distribuídos
Docente: Raimundo José de Araújo Macêdo
Discentes: Hérson Rezende e Kennedy Anderson
Semestre: 2023.2

Solução do projeto prático sobre Sistemas Bancários

A solução para o desafio proposto pelo professor irá depender, como um todo, do uso da biblioteca de *sockets*. Estes, são interfaces de programação de aplicativos (APIs) que permitem a comunicação entre computadores em uma rede.
No contexto deste desafio, optamos por adotar uma abordagem estruturada, organizando a solução do nosso sistema em três arquivos Python distintos: “utils.py”, “server.py”, e “client.py”. Essa estrutura modular visa melhorar a legibilidade, manutenção e escalabilidade do projeto, permitindo-nos desenvolver de forma mais eficiente e compreensível.

- “utils.py”: Este arquivo é dedicado a funcionalidades auxiliares que contêm definições e recursos compartilhados entre o servidor e os clientes. Aqui, centralizamos constantes, funções utilitárias e, se necessário, classes personalizadas que são relevantes para o sistema desenvolvido.
- “server.py”: Este é o componente central do nosso sistema e representa o servidor. Nele estão presentes todas as funcionalidades relacionadas à manutenção do estado das contas dos clientes e à execução das operações solicitadas. Isso inclui operações como depósitos, saques, transferências e consultas de saldo. O servidor também é responsável por lidar com a comunicação de rede, escutando as conexões dos clientes em um endereço IP e porta específicos e enviando respostas apropriadas.
- “client.py”: Neste arquivo, concentramos as funções necessárias para que os clientes se conectem ao servidor. Quando executado, simula um caixa eletrônico onde os clientes podem usar a interface para estabelecer comunicação com o servidor, através da troca de mensagens, e iniciar transações. Sendo essas transações as operações bancárias que foram definidas no servidor.

Essa abordagem estruturada, separando as funcionalidades em módulos distintos, facilita a manutenção do código, o trabalho em equipe e o desenvolvimento de recursos adicionais, caso necessário. Além disso, permite a reutilização de componentes compartilhados, como funções utilitárias, entre o servidor e os clientes.

Servidor

Com o objetivo de realizar as operações mencionadas anteriormente, desenvolvemos o servidor com um conjunto de métodos que desempenham funções cruciais no funcionamento do sistema. A seguir, detalhamos o fluxo de funcionamento do servidor:

1. Inicialização do Servidor: Ao iniciar o servidor, um procedimento é acionado para verificar se já existem contas cadastradas. Isso é alcançado através da leitura de dados armazenados, e se houver contas previamente registradas, o servidor carrega essas informações. Caso não existam contas cadastradas, o servidor cria contas de acordo com o que é especificado no módulo "utils.py". Este processo é fundamental para estabelecer o estado inicial do sistema financeiro simulado.
2. Aguardando Conexões de Clientes: Após a inicialização, o servidor entra em um estado de espera ativa, aguardando solicitações de conexão dos clientes. Para essa finalidade, utiliza-se o mecanismo de sockets para a comunicação, permitindo a troca de dados entre o servidor e os clientes.
3. Estabelecimento de Conexão: Quando um cliente solicita uma conexão, o servidor responde positivamente e inicia o processo de estabelecimento da conexão. Para lidar com múltiplos clientes simultaneamente, um novo thread é criado para cada cliente, permitindo que o servidor atenda várias solicitações concorrentemente.
4. Autenticação do Cliente: Antes de permitir que um cliente execute qualquer operação financeira, é essencial que o mesmo se autentique. Isso é feito o cliente fornecendo um identificador de conta, que neste contexto é representado como um número inteiro, que representa o RG do cliente. O servidor verifica se o cliente fornecido existe entre as contas cadastradas. Se a conta for encontrada, o servidor envia uma mensagem de confirmação ao

cliente, indicando que a autenticação foi bem-sucedida. Se a conta do cliente não for encontrada, o servidor emite uma mensagem para o cliente, notificando-o de que a conta não foi localizada e solicitando que tente novamente.

5. Execução de Operações: Após a autenticação bem-sucedida, o servidor aguarda o cliente escolher entre as diversas operações disponíveis, como depósitos, saques, transferências e consultas de saldo. O servidor executa as operações solicitadas pelo cliente e retorna as respostas apropriadas.

Cliente

Por outro lado, a perspectiva do cliente é dotada de um conjunto de métodos destinados a possibilitar a conexão com o servidor e a execução das operações desejadas. Aqui, destacamos de forma mais detalhada o fluxo de ações realizadas pelo cliente:

1. Estabelecimento da Conexão: Ao ser iniciado, o cliente inicia um método de conexão responsável por estabelecer a comunicação com o servidor. Isso é alcançado por meio do uso da biblioteca de sockets, onde o cliente especifica o IP e a porta pela qual deseja se conectar ao servidor. A tentativa de estabelecimento da conexão resulta em uma resposta, que é exibida para o usuário. Essa resposta é crítica para verificar se a conexão com o servidor foi bem-sucedida.
2. Autenticação do Cliente: Após a confirmação da conexão bem-sucedida, o cliente é solicitado a se autenticar. Esse processo requer que o cliente forneça um identificador de conta específico. Este identificador é fundamental para que o servidor possa reconhecer o cliente e validar sua identidade. A autenticação é concluída quando o cliente envia o identificador de conta ao servidor e aguarda a resposta correspondente. Caso a autenticação seja bem-sucedida, o cliente pode prosseguir.
3. Interatividade com o Sistema: Com a autenticação concluída, o cliente ganha acesso ao sistema e pode interagir com ele. Isso inclui a seleção de operações específicas, como depósitos, saques, transferências e consultas de saldo. O usuário fornece os dados necessários para executar a operação desejada. O módulo cliente troca mensagens com o servidor para informar a

operação e os dados necessários. O servidor, após receber as informações da operação e os dados do cliente, executa a operação correspondente. Em seguida, envia de volta ao cliente uma resposta, que é exibida para o usuário. O cliente pode, então, optar por executar outras operações disponíveis, ou desconectar, o que encerra a comunicação entre o cliente e o servidor.

Controle do Tempo

Nesta implementação foi utilizado o conceito de relógio de Lamport para garantir a ordenação adequada das operações de um ambiente distribuído entre cliente e servidor bancário. A implementação do relógio de Lamport envolve o uso de contadores ou carimbos de tempo lógicos para marcar eventos e estabelecer uma ordem parcial entre eles. Cada processo em um sistema distribuído mantém um contador local que é incrementado sempre que o processo executa um evento significativo. Esses eventos podem incluir a execução de uma operação importante, o envio ou a recepção de uma mensagem, entre outros.

No Cliente (client.py):

- O cliente mantém um contador local chamado ***time*** que é inicializado com valor 0 e que é incrementado sempre que o cliente executa um evento significativo (como enviar uma mensagem ou realizar uma operação bancária). O valor do contador local é incluído nas mensagens enviadas para o servidor.
- O cliente também possui a função ***ajustar_valor_relogio_logico()***, que recebe como parâmetro ***server_time,*** sendo o contador de tempo recebido no objeto de mensagem do servidor. Essa função irá ajustar o valor do relógio lógico do cliente com base no tempo do servidor, garantindo que o relógio do cliente esteja sempre sincronizado com o servidor.

No Servidor (server.py):

- O servidor também possui um contador local chamado ***time*** que é inicializado com valor 0.
- O servidor possui uma função ***incrementar_relogio()*** que é chamada sempre que uma nova conexão com um cliente é estabelecida e quando operações são realizadas.

- O método ***ajustar_relogio_timestamp(),*** que recebe como primeiro parâmetro o tempo do cliente e como segundo o timer do próprio servidor e, em seguida, ajusta o valor do relógio lógico do servidor pegando o maior tempo entre eles e incrementando em 1.
- No servidor, o tempo do cliente é considerado ao processar as operações. O tempo do cliente é incluído nas mensagens recebidas e é usado para ajustar o tempo do servidor, garantindo que as operações do cliente sejam tratadas na ordem correta. Para isso também é necessário utilizar o mutex, que trava atomicamente uma operação para que ela não seja interrompida por outra.