Projeto 1
Programação de Redes de Computadores

João Paulo Penalber RA: 170556
Lucas Cunha Agustini - RA: 172655

I. Introdução

Neste projeto implementamos um servidor para catalogar filmes em um cinema. Para esta primeira iteração o servidor foi implementado utilizando o protocolo TCP, através da linguagem C e *syscalls* do sistema Linux, para armazenar os dados foi escolhido o banco sqlite.

II. Execução

- make all
- inicializar o banco com:
    - ./bin/resetdb.x
- rodar o servidor e o cliente
    - ./bin/server.x
    - ./bin/client.x
- no cliente escolher uma das ações citadas em Descrição.

III. Descrição

O servidor deve armazenar os seguintes dados de cada filme:

- Identificador único
- Nome
- Sinopse
- Gênero
- Sala em exibição

Além de armazenar estes dados, o servidor também expõe uma interface para acessar e inserir filmes. A interface é utilizada através de uma aplicação cliente que disponibiliza as seguintes funções para o usuário:

- **inserir**: insere um novo filme na base de dados
- **listar_titulo**: lista os nomes e salas de todos os filmes
- **listar_genero**: lista os nomes dos filmes de um dado gênero
- **nome**: retorna o nome do filme a partir do identificador
- **info**: retorna todas as informações de um dado identificador
- **tudo**: lista todas as informações de todos os filmes

O banco de dados roda localmente na mesma máquina do servidor, dessa forma um programa para reset, que instancia e preenche o banco, foi criado.

IV. Implementação

O banco de dados possui duas entidades, Filme e Sala, e um relacionamento, Exibição, de forma que Exibição liga vários Filmes em várias Salas. O banco inicialmente já está populado com 3 filmes, 3 Salas e 3 Exibições, tais configurações iniciais podem ser modificadas no *resetdb.c*. Atualmente a única forma de adicionar Salas novas é no reset do banco, ou diretamente no console do SQLite.

Foi implementado uma interface com o banco de dados para o servidor realizar as queries desejadas ao banco através da biblioteca C do SQLite.

Tanto o cliente quanto o servidor utilizam *syscalls* do kernel Linux para criar um *socket* e se conectar via TCP.

Para se comunicar, foi definido uma estrutura uniforme, cujo tipo de chamada ou retorno é definido através de uma enumeração compartilhada pelos cliente e servidor e os dados vem em seguida em um *buffer*, que é interpretado de acordo com a numeração recebida.

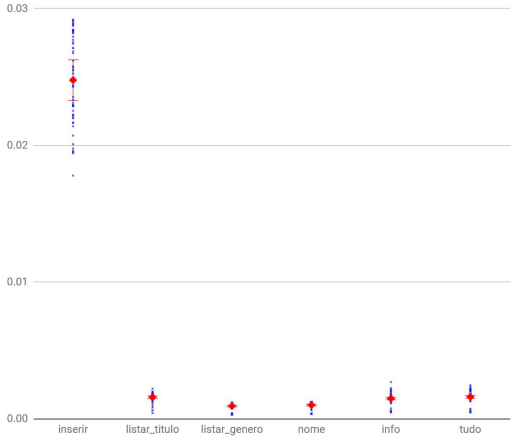
V. Resultados

Para avaliar as aplicações resultantes deste trabalho instrumentamos o cliente e servidor de modo a obter o tempo total do cliente fazer a chamada e receber a resposta e o tempo que o servidor leva para processar a chamada recebida.

O ambiente de teste é composta de *scripts* que executam o cliente e servidor, simulando 30 chamadas de cada tipo. O cliente e servidor estavam em máquinas e redes diferentes, se comunicando através da internet.

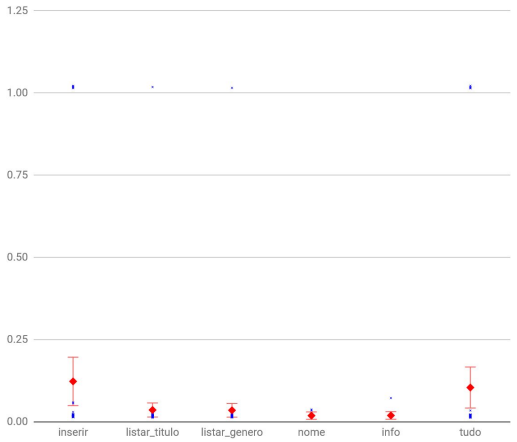
Na figura 1 temos o tempo de processamento de 30 chamadas de cada um dos tipos que o servidor disponibiliza. Podemos ver que o processamento é bastante veloz em todos os casos, apesar da inserção ser mensuravelmente mais lenta que o restante das operações.

Figura 1: Tempo de processamento (servidor)



Ao subtrair o tempo que o cliente demorou para receber a resposta da chamada do tempo de processamento do servidor obtemos o tempo de comunicação do experimento.

Figura 2: Tempo de comunicação



Na figura 2 temos estes tempos, e é possível ver que as operações que transmitem mais dados levam mais tempo que o restante, e também que o tempo de comunicação é a maior parte do tempo que o cliente espera pela resposta.

VI. Conclusão

Podemos ver que o servidor e cliente se comunicam sem problemas, mesmo com muitos pedidos simultâneos como o *script* de teste demonstrou.

Os testes demonstraram que o gargalo na resposta está na comunicação pela rede, porém ignorando isso, temos que o acesso ao banco de dados é a segunda causa de possível lentidão, apesar de estarmos lidando com uma escala de tempo muito pequena na análise.